Universidad Carlos III de Madrid
Escuela Politécnica Superior

Ingeniería Técnica de Telecomunicaciones: especialidad en Sonido e Imagen

Proyecto Fin de Carrera

UPnP Media Renderer para plataforma OSGi

Autor: Francisco Mota Porta
Tutor: Julio Ángel Cano Romero

Julio de 2009

Agradecimientos

Este proyecto fin de carrera supone el fin de una etapa en mi vida, y el comienzo de otra aún por descubrir. Es momento por tanto para la reflexión, echar un vistazo atrás y recuperar aquellos momentos que han marcado huella. Entre todos ellos aparecen los vividos con compañeros, profesores y personal de la universidad. Tengo pues que agradecerles a todos ellos el haberme permitido llegar hasta aquí. En especial, a aquellas personas que con su labor y actitud de proximidad al otro hacen que el trayecto sea más ameno y el aprendizaje más interesante.

Mi primer agradecimiento personalizado es para mi tutor, Julio, por su atención y dedicación durante todo el proceso de elaboración del presente proyecto. Gracias a sus compañeros de despacho, Javier y Mario, por sus prácticos consejos. Y también a Natividad por habérmelo ofrecido.

Gracias a mis padres, a mis hermanas y a sus respectivas parejas por el respeto, apoyo y cariño que demuestran en todo lo que hago, y en especial en lo que respecta a este proyecto.

A todos mis amigos, gracias por su cariño y comprensión durante todo este tiempo que he estado sumergido en esta labor.

Por último, quiero hacer un agradecimiento especial a dos amigas, Irene y Marina, que se han convertido en cómplices de mi tiempo en esta universidad y con las que he compartido muchos y muy buenos momentos.

Resumen

El aumento del equipamiento electrónico de los hogares provoca que las compañías tecnológicas (informática, audiovisuales, telecomunicaciones,...) colaboren unas con otras con el objetivo de crear nuevos estándares que permitan la interoperabilidad entre estos dispositivos.

Universal Plug and Play (UPnP) es un ejemplo de estos estándares, promovido por el *UPnP Forum*. Esta tecnología define un conjunto de protocolos comunes basados en Internet que los dispositivos pueden utilizar para agregarse a una red y describirse a sí mismos y sus capacidades. Esto permite que otros dispositivos y personas puedan usarlos sin tener que realizar ninguna configuración. Esta interoperabilidad entre diferentes tipos de dispositivos puede resultar realmente útil para nuevos sistemas multimedia integrados en los entornos de red del hogar. En este sentido, el Foro UPnP especifica una arquitectura para dispositivos audiovisuales, la *UPnP AV Architecture*. Esta arquitectura define la interacción entre un Punto de Control UPnP para audio y video (*UPnP AV Control Point*) y un dispositivo UPnP para audio y video (*UPnP AV Device*).

Un *UPnP AV Control Point* maneja y coordina la comunicación entre un dispositivo servidor de contenido multimedia, denominado *MediaServer*, y un dispositivo consumidor de contenido multimedia, denominado *MediaRenderer*. Esta comunicación permite a un usuario compartir todo el contenido multimedia localizado por los dispositivos *UPnP MediaServer* y reproducirlo en cualquier dispositivo *UPnP MediaRenderer* que esté agregado a su red UPnP.

A parte de esto, el incremento del uso de dispositivos avanzados como las Pasarelas de Servicios, para combinar las diferentes redes domésticas y conectarlas con el exterior (a través de Internet), estimula el desarrollo software de aplicaciones y servicios para ser ejecutados en estas pasarelas. La tecnología OSGi proporciona una plataforma software para el desarrollo de este tipo de aplicaciones y servicios.

Este proyecto presenta una solución para un dispositivo software *UPnP MediaRenderer* conforme a una aplicación OSGi.

Abstract

Electronic home equipment increase forces to the technological companies (informatics, audiovisuals, telecommunications, etc) to collaborate ones each others with the purpose of making new standards for interoperability among devices.

Universal Plug and Play (UPnP) is an example of these standards. It is promoted by the UPnP Forum. This technology defines a set of common Internet-based protocols that devices can use to join a network and describe themselves and their capabilities, which enables other devices and people to use them without setup or configuration. This interoperability among different kinds of devices can be really useful for new multimedia systems in home network environments. In this regard, the UPnP Forum specifies an architecture for audiovisual devices, the UPnP AV Architecture. This architecture defines the general interactions between an UPnP AV Control Point and an UPnP AV Device.

An UPnP AV Control Point manages and coordinates the communication between a source device of media content (called MediaServer), and a sink device for that content (called MediaRenderer). This communication allows users to share all available media content located on UPnP MediaServers and to render it on whatever UPnP MediaRenderer joined to their UPnP network.

A part from that, the increase use of advanced devices like Service Gateways to combine the different domestic networks and interconnects them with the outside (thought Internet), promotes software development of applications and services to be executed in this gateways. The OSGi technology provides a software platform to develop this kind of applications and services.

This project introduces a solution for an UPnP MediaRenderer software device according to an OSGi application.

Índice

ANEXOS

Índice de figuras

Índice de tablas

Índice de cuadros de texto

1. Introducción

Este capítulo sirve de introducción a este proyecto fin de carrera. Primeramente se expondrán las motivaciones que impulsan su realización, presentando los conceptos base del mismo. Posteriomente, se enumerarán los objetivos marcados para este proceso. Y en último lugar, se describirá la estructura del presente documento.

1.1 Motivación

La evolución de los hogares hacia la integración de sistemas informáticos provoca la proliferación de múltiples equipos y sistemas electrónicos que ofrecen distintos servicios con el propósito de facilitar tareas y entretener al usuario de éstos. Sin embargo, hasta hace poco estos dispositivos permanecían aislados sin posibilidad de compartir sus servicios o poder utilizar los de otros dispositivos. En este sentido, se hace necesario proporcionar tecnologías que permitan conectarlos entre sí y con el exterior a través de la red doméstica, generando servicios más avanzados y facilitando la gestión de los mismos. De este forma, se sientan las bases de lo que se ha venido a definir como el Hogar Digital.

Existen varias iniciativas que pretenden dar soluciones a la conectividad entre dispositivos, proporcionando tecnologías de comunicación diferentes y en muchos casos incompatibles. Ejemplo de ellas son: HomePNA, HomePlug, x10, LonWorks, HAVi (*Home Audio Video Interoperability*), JINI, etc.

Con el mismo propósito surgió la tecnología UPnP (*Universal Plug and Play*), la cual define un arquitectura para el descubrimiento automático y dinámico de dispositivos, y el intercambio de información y datos entre los mismos. Está impulsada por el *UPnP Forum* y basada en protocolos estándar de Internet como IP, TCP, UDP, HTTP y XML.

En lo que se refiere a sistemas de entretenimiento digital, más allá de la comunicación, resulta tedioso que en la mayoría de estos sistemas, para poder reproducir el contenido multimedia almacenado en un equipo se deba acudir al correspondiente reproductor instalado en el mismo. Para ello, UPnP especifica una arquitectura, *UPnP AV Architecture*, con la que el contenido multimedia localizado por un dispositivo puede ser compartido en la red UPnP y ser reproducido remotamente en dispositivos reproductores distribuidos por esta. A los dispositivos que proveen el contenido multimedia se les denomina *UPnP MediaServer*, mientras que los que los consumen son llamados *UPnP MediaRenderer*. La comunicación entre ambas entidades se realiza por medio de una tercera entidad coordinadora denomina Punto de Control para Audio y Video (en Inglés *AV Control Point*), que además controla el proceso de reproducción de forma remota.

Al hilo de esto, parece que el futuro, ya muy presente, del Hogar Digital pasa por la utilización de un elemento que integre las distintas redes domésticas (HomePAN, LonWorks, HAVi, UPnP,...) y las interconecte con el exterior. Este elemento es conocido como Pasarela Residencial, también como Pasarela Doméstica, o de forma más general Pasarela de Servicios. La Pasarela Residencial conecta las infraestructuras de telecomunicaciones (datos, control, automatización, etc.) de la vivienda a una red pública de datos, como por ejemplo Internet, y sirve de plataforma de ejecución para las aplicaciones domésticas, todo ello de manera transparente para el usuario.

Debido a la diversidad de tecnologías existentes en los hogares, se hace evidente la necesidad de un estándar común que establezca un marco de trabajo sobre el que las compañías puedan desarrollar servicios y aplicaciones para las pasarelas, sin tener que preocuparse por aspectos de bajo nivel como tecnologías de red o hardware. En este ámbito de necesidad nace OSGi (*Open Services Gateway Initiative*) como estándar que define una arquitectura software que se ejecuta en una pasarela residencial.

En consecuencia, toda aplicación que se desarrolle para su utilización en redes domésticas debe buscar la compatibilidad con tecnologías como OSGi.

Este Proyecto Fin de Carrera trata sobre el desarrollo de una aplicación Java que conforma un dispositivo UPnP para la reproducción de contenido multimedia, de acuerdo a la especificación UPnP para dispositivos de audio y video, y para su ejecución en plataformas de servicios OSGi como las pasarelas residenciales.

1.2 Objetivos del proyecto

El objetivo final de este proyecto es el desarrollo de un prototipo de dispositivo *UPnP MediaRenderer* para su integración en plataformas OSGi con las siguientes características generales:

- Programación Java: la aplicación habrá de estar programada en Java.
- Multiplataforma: se deberá buscar una solución que pueda ser utilizada tanto en plataformas Windows como Linux.
- Control de la renderización en remoto conforme al protocolo UPnP.
- Compatibilidad IPv6: tendrá que poder ser desplegada en escenarios conformes al protocolo de red de nueva generación IPv6.
- Renderización de flujos HTTP, RTP e imágenes: será capaz de reproducir contenidos de audio y video obtenidos mediante flujos HTTP o RTP, así como visualizar imágenes.
- Soporte a los formatos más comunes: deberá procesar el contenido con los formatos más comunes de codificación ya sea audio, video o imágenes.

Para alcanzar este objetivo se definen los siguientes subobjetivos:

1. Estudiar las tecnologías implicadas para el diseño del prototipo:
    a. Estudio de la tecnología UPnP prestando mayor atención en la especificación para un *UPnP MediaRenderer*. En este estudio se incluirá un análisis del estado del arte de este tipo de dispositivos.
    b. Estudio de la librería Java que se utilizará para proporcionar la capacidad de reproducción multimedia, *Java Media Framework* (JMF).
    c. Estudio de la tecnología OSGi para el desarrollo de aplicaciones que se integren en escenarios gestionados por pasarelas de servicios.
2. Analizar los requisitos del sistema en base a las características generales mencionadas anteriormente.
3. Construir un prototipo de *UPnP MediaRenderer*: a partir de los conocimientos obtenidos en los objetivos previos se podrá proceder a la implementación de la aplicación.
4. Asegurar el funcionamiento del prototipo en un escenario real y definir las limitaciones derivadas de su diseño.

1.3 Contenido de la memoria

La presente memoria está estructurada en cinco capítulos que resumen todo el proceso para la consecución de los objetivos anteriormente planteados. Además se adjuntan ocho anexos.

Este primer capítulo sirve de introducción al proyecto, exponiendo los motivos de su realización, así como los objetivos que se persiguen.

El segundo capítulo conforma el marco teórico del proyecto. Se incluyen sendos apartados dedicados a cada una de las tecnologías sobre las que se fundamenta el desarrollo de la aplicación: UPnP, JMF y OSGi. Al final del capítulo se incluye un apartado sobre el estado del arte, en el que se hará un breve análisis de las soluciones de dispositivos *UPnP MediaRenderer* actuales, se evaluará la situación de la biblioteca multimedia JMF, planteando diferentes alternativas que resuelvan sus limitaciones, y se enunciarán las principales implementaciones software de la plataforma OSGi.

El tercer capítulo se hace el estudio sobre la implementación del sistema. Primeramente se analizarán los requisitos del mismo definiendo sus casos de uso, y posteriormente se enunciarán las herramientas que han sido utilizadas en su desarrollo. Por último se presentará el prototipo diseñado definiendo los módulos en los que se compone, así como los paquetes y clases que lo componen. Se

utilizarán diagramas de secuencia de aquellos procesos que definen las principales funcionalidades de la aplicación.

El cuarto capítulo recoge las pruebas realizadas al sistema en un entorno real de ejecución, y en las que se muestran la eficiencia de la aplicación en los procesos más importantes.

En el quinto y último capítulo se extraerán las conclusiones del proyecto, se describirán las limitaciones detectadas del prototipo y se plantean las posibles líneas de trabajo futuras para el mismo.

Seguidamente se incluye una sección de anexos. Unos para ampliar la información contenida en la memoria y otros con los documentos que no tenían un lugar definido en la misma. Entre ellos se encuentra el presupuesto del proyecto, el manual de instalación de la aplicación, y los ficheros de configuración y descripción del dispositivo.

2. Marco teórico

A lo largo de este capítulo se presentarán las bases teóricas sobre las que se asienta este proyecto. Tras dar una visión global de las tecnologías utilizadas en su desarrollo (UPnP, JMF y OSGi), se realizará un estudio más detallado de aquellos aspectos que resulten más importantes para la implementación de un prototipo de dispositivo *UPnP MediaRenderer* y se analizará su presencia en el mercado.

2.1 Universal Plug and Play (UPnP)

En los próximos subapartados se estudiará la tecnología UPnP. Se expondrán las bases conceptuales de ésta para posteriormente describir la arquitectura que define, así como los protocolos sobre los que está basada y las distintas fases de una comunicación UPnP. Posteriormente se presentará la arquitectura que define esta tecnología para dispositivos audiovisuales, y se enumerarán las limitaciones que presenta. Finalmente se justificará su uso en este proyecto.

2.1.1 Introducción

UPnP, de las siglas en inglés *Universal Plug and Play*, es un protocolo de transmisión de datos punto a punto para la comunicación entre aplicaciones. Define una arquitectura software, abierta y distribuida, que proporciona una forma de conectividad entre distintos equipos pertenecientes a una red, permitiendo el intercambio de información y datos entre los mismos [UPNPCASADOMO].

Surge a partir del modelo *Plug and Play* para la conexión de periféricos al PC (*Personal Computer*), cuya máxima, "enchufar y listo para usar", es extendida a dispositivos inteligentes, dispositivos móviles, PCs, electrónica de consumo, y aplicaciones software.

Usando el modelo UPnP, un dispositivo puede agregarse dinámicamente a una red, obtener una dirección IP, anunciar sus servicios, y saber de la existencia de otros dispositivos, todo esto de forma automática y transparente para el usuario final.

Varios son los escenarios en los que se puede implementar. Ejemplos de estos son la automatización del hogar, impresoras en red, electrodomésticos para la cocina, o el entretenimiento digital, como es el caso de este proyecto.

Según el Foro UPnP [UPNPFORUM] esta tecnología aporta beneficios tales como:

- **Independencia de medios (tipos de redes) y dispositivos:** Puede funcionar sobre cualquier medio que soporte el protocolo IP, incluyendo líneas telefónicas o eléctricas, Ethernet, FireWare, Infrarrojos(IR), radio (RF): Wi-Fi, Bluetooh, etc. Esto hace que su uso sea apropiado para la Domótica.

- **Independencia de la plataforma y del lenguaje de programación:** Se puede usar sobre cualquier sistema operativo y utilizando cualquier lenguaje de programación para crear productos UPnP. De hecho, UPnP no especifica ninguna API (*Aplication Programming Interface*) para aplicaciones, dando libertad a los desarrolladores y fabricantes de utilizar sistemas operativos y capacidades acordes a las necesidades de sus clientes.

- **Tecnologías basadas en Internet:** Está desarrollada sobre protocolos comunes de Internet tales como IP (*Internet Protocol*), TCP (*Transmission Control Protocol*), UDP (*User Datagram Protocol*), HTTP (*Hypertext Protocol*) y XML (*Extensible Markup Language*)

- **Interfaz de usuario de control del dispositivo:** La arquitectura UPnP permite presentar una interfaz de usuario en un explorador web a través de la cual un usuario

final podría, dependiendo de las capacidades del dispositivo, controlar a éste en remoto.

- **Control por programa de la aplicación:** Un programa puede controlar directamente un dispositivo UPnP sin necesidad de que éste posea un interfaz de usuario. Este control puede basarse en la información descubierta sobre el dispositivo o a través de un conocimiento a priori del tipo de dispositivo.

- **Protocolos base comunes:** Esta tecnología se fundamenta en protocolos y procedimientos comunes, con los que los desarrolladores de aplicaciones y dispositivos están de acuerdo, asegurando la interoperatividad entre los múltiples dispositivos existentes en el mercado.

- **Extensibilidad:** Los fabricantes o desarrolladores pueden introducir servicios de valor añadido a sus productos UPnP, sobre una arquitectura básica común.

El Foro UPnP es una asociación de compañías y personas relacionadas con la industria del sector tecnológico, formada en Octubre de 1999, cuyo propósito es promover el desarrollo de dispositivos de fácil conexión y simplificar su implementación en redes del hogar y entornos empresariales. Esto lo logra definiendo y publicando descripciones de Dispositivos y Servicios UPnP, originalmente denominadas DCPs (*Device Control Protocols*), de acuerdo a una arquitectura común de dispositivo impulsada por Microsoft. [UPNPFORUM]

2.1.2 Componentes de una red UPnP

La arquitectura UPnP define dos nodos principales de comunicación: los Puntos de Control y los Dispositivos.

Dispositivos (Device) y Servicios

Un dispositivo UPnP puede ser tanto un dispositivo real (hardware), como una aplicación software. Se trata de un contenedor de servicios y/o otros dispositivos. Estos servicios y dispositivos embebidos tienen como objetivo definir la funcionalidad del dispositivo que los contiene.

Poniendo como ejemplo de dispositivo contenedor un televisor LCD con reproductor DVD integrado. Los servicios que contiene podrían ser: un servicio de control de la visualización, un servicio de sintonización, y un servicio de reloj. A su vez tiene un dispositivo embebido, el reproductor DVD, con sus propios servicios asociados, como por ejemplo: un servicio de control de la reproducción, un servicio de afinamiento, etc. El conjunto de los servicios y dispositivos integrados define las características funcionales del televisor combo.

Toda esta información se recoge en un documento XML de descripción, DCPD (*Device Control Protocol Document*), que el propio dispositivo debe alojar.

Los servicios conforman la unidad más pequeña de control en una red UPnP. Proporciona un conjunto de acciones y modela su estado con variables de estado. Siguiendo con el ejemplo del televisor LCD, una variable de estado del servicio de reloj podría ser la hora actual, y una acción sería obtener dicha hora.

De forma análoga a la descripción del dispositivo, esta información queda reflejada en un documento XML de descripción del servicio denominado SCPD (*Service Control Protocol Definition*).

Además, los cambios de estas variables de estado podrían generar eventos que las entidades subscritas para escucharlos, comúnmente los Puntos de Controls, pueden utilizar para mantener actualizada la información de estado del dispositivo controlado.

Puntos de Control (Control Point)

Los puntos de control UPnP son controladores de dispositivos UPnP, capaces de descubrir y comunicarse con ellos. Después de escuchar los mensajes de descubrimiento de los dispositivos, pueden obtener la descripción de los mismos con la lista de servicios asociados y sus correspondientes descriptores, invocar acciones para controlar su comportamiento, e incluso subscribirse a la fuente de eventos de estos para poder enterarse de los cambios en sus variables de estado.

En la siguiente figura se muestra la relación entre los servicios de los dispositivos y los puntos de control:

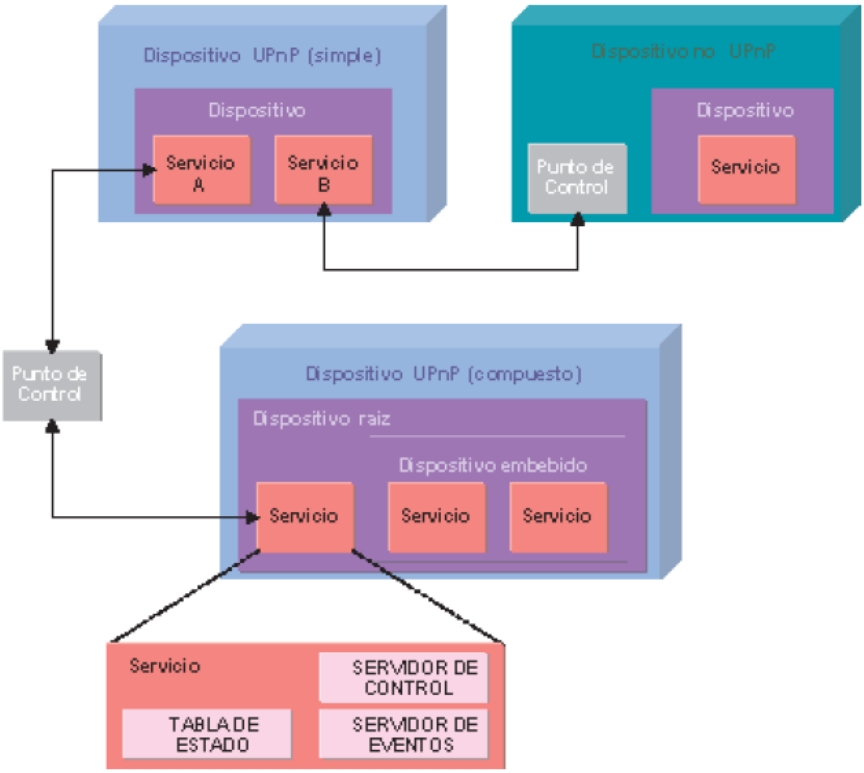


Figura 2.1. Esquema Puntos de Control, Dispositivos y Servicios [AUNADOC]

Puentes (Bridge)

Los puentes UPnP, son entidades de enlace entre la red UPnP y dispositivos que no pueden pertenecer a ésta por carecer de recursos o por no soportar protocolos como TCP/IP, HTTP o XML. Un ejemplo de escenario con un UPnP *Bridge* sería el definido en la siguiente figura.

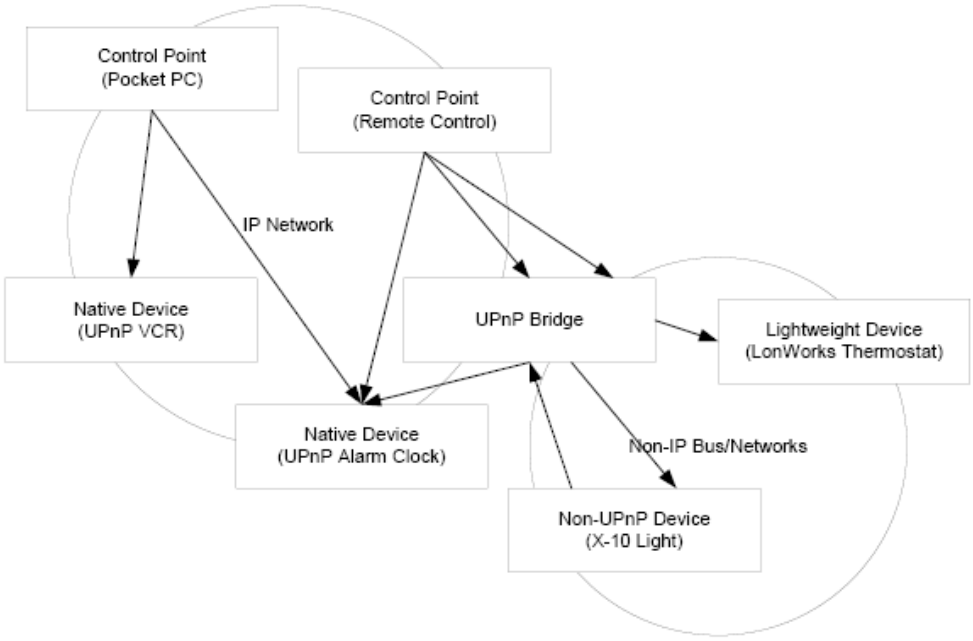


Figura 2.2. Ejemplo componentes UPnP [UPNPPROT]

2.1.3 Protocolos usados por UPnP

Como se comentó anteriormente, uno de los principales beneficios de UPnP es que utiliza protocolos estándares basados en Internet, comúnmente utilizados tanto en la propia red Internet como en redes de área local. Esto, a parte de asegurar el conocimiento de los mismos por la mayor parte de desarrolladores de dispositivos, permite establecer la independencia de la arquitectura respecto del medio donde se implemente.

A continuación se detallan los protocolos en los que se fundamenta la tecnología UPnP [UPnPPROT]:

TCP/IP

Familia de protocolos de Internet sobre el que se asientan el resto de protocolos utilizados por UPnP y que permiten la conectividad a nivel de red entre los dispositivos. Con su uso se asegura la anteriormente mencionada interoperabilidad en diferentes medios físicos e independencia de las plataformas.

Dentro de la pila de protocolos TCP/IP, se hace uso de los más comunes. Ejemplo de estos son: IP (*Internet Protocol*) como protocolo de red, TCP (*Transmission Control Protocol*) como protocolo fiable de transporte, o UDP (*User Datagram Protocol*) como protocolo no fiable de transporte.

HTTP, HTTPU, HTTPMU

HTTP supone la auténtica base sobre la cual se desarrolla toda la tecnología UPnP. No en vano, permite la comunicación al nivel de aplicación entre los dispositivos que conforman la red. HTTPU y HTTPMU son variantes de HTTP utilizados para entregar mensajes a través de UDP/IP. En el primer caso de un dispositivo a otro (*unicast*), y en el segundo, de un dispositivo al resto que están disponibles en la red UPnP (*multicast*).

SSDP (Simple Service Discovery Protocol)

Como su propio nombre indica, define cómo unos servicios de red pueden ser descubiertos por otras entidades pertenecientes a la misma red utilizando direccionamiento

multicast. Define cómo puede un Punto de Control localizar recursos, dispositivos, de interés en la red, y como puede un dispositivo anunciar su presencia en la misma. UPnP lo utiliza para la fase de "Descubrimiento" de dispositivos.

Toma como protocolos base HTTPMU y HTTPU. El primero es utilizado para el envío de mensajes de búsqueda de dispositivos (utilizado por el Punto de Control) o anuncios de descubrimiento (utilizado por los dispositivos). El segundo se usa para el envío de los correpondentes mensajes de respuesta de los dispositivos a los mensajes de búsqueda.

GENA (General Event Notification Architecture)

Provee una arquitectura para el envío y recepción de mensajes de notificación utilizando HTTP sobre TCP/IP y multidifusión sobre UDP (HTTPMU). A su vez, define los conceptos de subscriptores y editores de notificaciones.

En UPnP, los formatos GENA son usado para los anuncios de presencia enviados a través de SSDP comentados anteriormente, así como para proveer la capacidad de informar de los cambios en los estados de los servicios a través de eventos. Por tanto, UPnP la utiliza para la fase de "Envío y recepción de eventos", y en la fase de "Descubrimiento".

SOAP (Simple Object Access Protocol)

Define el uso de XML y HTTP para ejecutar llamadas de procedimiento remoto en entornos descentralizados y distribuidos. Se ha convertido en el estándar para la comunicación basada en la tecnología RPC (*Remote Procedure Call*) en Internet. Puede hacer uso de protocolos de seguridad como SSL (*Secure Socket Layer*) para comunicaciones más seguras.

UPnP utiliza este protocolo en la fase de "Control" de los dispositivos.

XML (Extensible Markup Language)

Es el formato universal para datos estructurados en la Web. Dicho de otra forma, es una forma de colocar cualquier tipo de datos estructurados, como los presentados en páginas Web, en un fichero de texto. El uso de XML como un lenguaje esquema, está definido por W3C.

UPnP lo utiliza para las descripciones de los dispositivos y servicios, y para los mensajes de control y eventos.

2.1.4 Fases de la comunicación UPnP

Como se ha comentado anteriormente, la tecnología UPnP proporciona la capacidad de comunicación entre Puntos de Control y Dispositivos. Para ello, y como consecuencia del uso de protocolos basados en Internet, define un conjunto de servidores HTTP para poder manejar las distintas fases de la comunicación.

Son seis las fases o mecanismos posibles que pueden darse durante la comunicación, Direccionamiento, Descubrimiento, Descripción, Control, Eventos y Presentación. [UPNPDEVARCH]

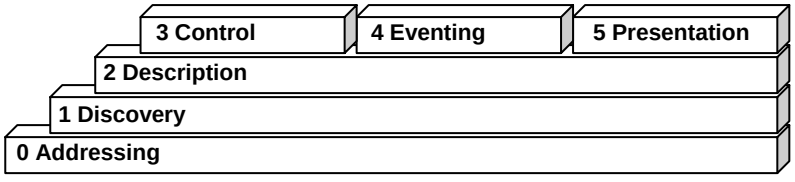


Figura 2.3. Esquema de las fases de la comunicación UPnP [UPNPEINDHOVEN]

Direccionamiento

Más que una fase de la comunicación UPnP, el Direccionamiento, se considera el paso previo al desarrollo de la misma. Se trata de que los dispositivos que integren la red UPnP obtengan una dirección IP con la que se identifiquen y puedan comunicarse con el resto de miembros. Para ello, el dispositivo debe disponer de un cliente *Dynamic Host Configuration Protocol* (DHCP) y buscar un servidor DHCP disponible en la red que le asigne una IP. De no encontrar ningún servidor, debe utilizar la función de *Auto-IP*, por la que obtiene una IP por él mismo.

Descubrimiento

Una vez que el dispositivo tiene asignada una IP única para conectarse a la red, está en condiciones de pasar a la primera de las fases de la comunicación UPnP, el Descubrimiento.

Para esta fase se hace uso del mencionado protocolo SSDP. Mediante este portocolo el dispositivo se agrega a la red. Este proceso le permite anunciar sus servicios a los Puntos de Control u otros dispositivos. A su vez, a un Punto de Control, le permite la búsqueda de dispositivos en la red.

En los mensajes enviados por el dispositivo, tanto de descubrimiento como los de respuesta, contienen información sobre algunos aspectos esenciales e importantes de los dispositivos y servicios que anuncia. Cabe destacar, la localización, en formato *Uniform Resource Locator* (URL), del documento de descripción del dispositivo o descriptor.

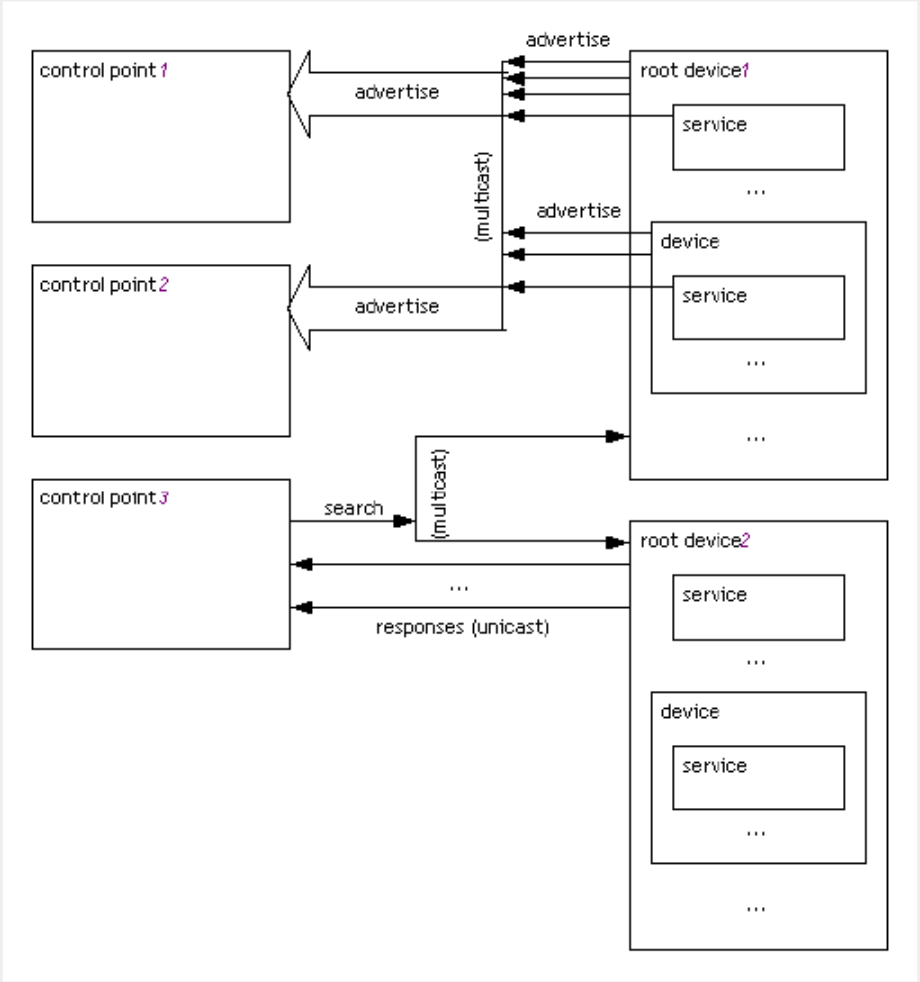


Figura 2.4. Esquema de la fase de Descubrimiento UPnP [UPNPDEVARCH]

Descripción

Después de que el Punto de Control ya haya descubierto al dispositivo, para que pueda conocer las capacidades de este, sus servicios y sus dispositivos embebidos, así como poder interactuar con él, debe obtener su descripción. Para ello debe recuperar el documento con esta descripción, el DCPD, a partir de la URL que, como se comentó en el apartado anterior, se especifica en el mensaje de descubrimiento.

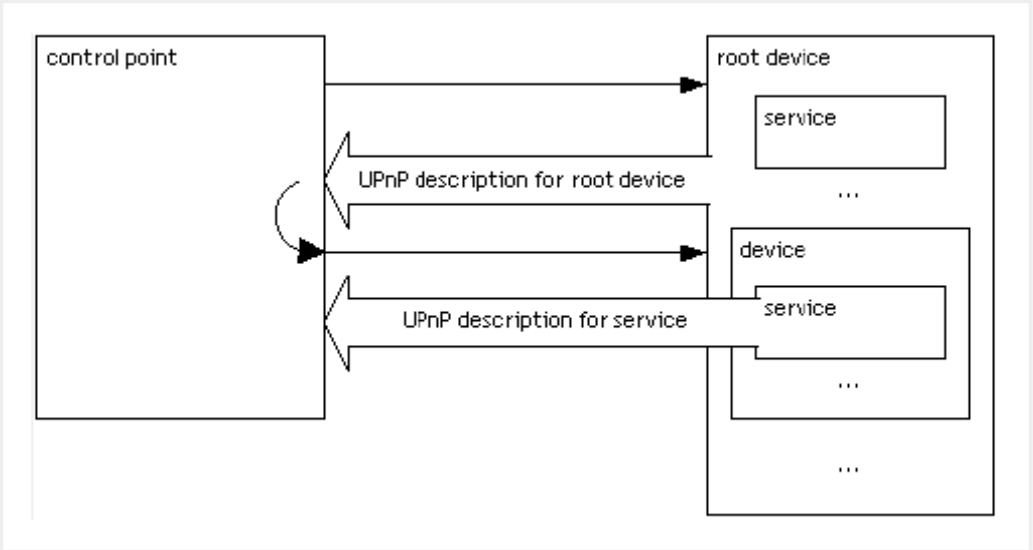


Figura 2.5. Esquema de la fase de Descripción de UPnP [UPNPDEVARCH]

La descripción completa del dispositivo está dividida en dos partes lógicas, la descripción del dispositivo y las descripciones de los correspondientes servicios asociados a éste.

Documento de descripción del dispositivo (DCPD)

Contiene información acerca del fabricante del dispositivo, información de fabricación, como nombre y tipo de dispositivo, numero de serie, urls del fabricante, etc. En el mismo documento se listan los distintos tipos de servicios que implementa el dispositivo. Para cada uno se incluye datos tales como, el nombre, una URL para su descripción, una URL para el mecanismo de Control, y una URL para el mecanismo de Eventos. Además, incluye una descripción de todos los dispositivos embebidos que contiene el dispositivo raíz, y una URL de presentación del dispositivo (en el apartado de Presentación se describirá su utilidad).

En el siguiente cuadro de texto se muestra el documente XML tipo para la descripción de un dispositivo:

```
<?xml version="1.0" encoding="utf-8"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <URLBase>URL base para todas las urls relativas</URLBase>
  <device>
    <deviceType>urn:schemas-upnp-org:device:TipoDeDispositivo:version</deviceType>
    <friendlyName>nombre corto</friendlyName>
    <manufacturer>nombre del fabricante</manufacturer>
    <manufacturerURL>url del fabricante</manufacturerURL>
    <modelDescription>nombre largo y descriptivo del dispositivo</modelDescription>
    <modelName>nombre del modelo</modelName>
    <modelNumber>numero del modelo</modelNumber>
    <modelURL>url al sitio del modelo</modelURL>
    <serialNumber>numero de serie del fabricante</serialNumber>
    <UDN>identificador unico y universal del dispositivo</UDN>
    <UPC>codigo universal del producto</UPC>
    <iconList>
      <icon>
        <mimetype>formato imagen</mimetype>
        <width>pixeles horizontales</width>
        <height>pixeles verticales</height>
        <depth>profundidad del color</depth>
        <url>URL de localizacion del icono</url>
      </icon>
        ...
    </iconlist>
    <serviceList>
      <service>
        <serviceType>urn:schemas-upnp-org:service:TipoServicio:version</serviceType>
        <serviceId>urn:upnp-org:serviceId:IDServicio</serviceId>
        <SCPDURL>URL para la descripcion del servicio</SCPDURL>
        <controlURL>URL para control</controlURL>
        <eventSubURL>URL para eventos</eventSubURL>
      </service>
        ... otros servicios definidos por el Foro UPnP
        ... otros servicios de valor añadido creados por el fabricante
    </serviceList>
    <deviceList>
      ... otros dispositivos embebidos definidos por el Foro UPnP
      ... otros dispositivos embebidos añadidos por el fabricante
    </deviceList>
  </device>
</root>
```

Cuadro de texto 2.1. XML de descripción de un dispositivo

Documento descriptor del servicio (SCPD)

En este documento se listan las variables de estado que definen el estado del servicio, así como el conjunto de acciones, con sus respectivos argumentos de entrada o salida, que están disponibles para ser invocados por el controlador, el Puntos de Control.

A continuación se muestra el documento XML tipo para la descripción de un servicio:

```
<?xml version="1.0" encoding="utf-8"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0>
   <specVersion>
      <major>1</major>
      <minor>0</minor>
   </specVersion>
   <serviceStateTable>
      <stateVariable sendEvents="yes"> (yes: si se envía evento cuando cambia)
         <name>nombre variable</name>
         <dataType>tipo de datos</dataType>
         <allowedValueList>
            <allowedValue>valor enumerado</allowedValue>
               ... otros valores definidos por el Foro UPnP
         </allowedValueList>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>nombre variable</name>
         <dataType>tipo de datos</dataType>
         <defaultValue>valor por defecto</defaultValue>
         <allowedValueRange>
            <minimum>valor minimo permitido</minimum>
            <maximum>valor maximo permitido</maximum>
            <step>incremento</step>
         </allowedValueRange>
      </stateVariable>
         ... otras variables de estado definidas por el Foro UPnP
         ... otras variables de estado definidas por el fabricante
   </serviceStateTable>
   <actionList>
      <action>
         <name>nombre accion</name>
         <argumentList>
            <argument>
               <name>nombre parametro</name>
               <direction>direccion: entrada (in) o salida (out)</direction>
               <relatedStateVariable>variable de estado relacionada</relatedStateVariable>
            </argument>
               ... otros argumentos definidos por el Foro UPnP
      </action>
         ... otras acciones definidas por el Foro UPnP
         ... otras acciones definidas por el fabricante
   </actionList>
</scpd>
```

Cuadro de texto 2.2. XML de descripción de un servicio

Como se mencionó anteriormente y a la vista de las figuras, se tratan de documentos XML, escritos por el fabricante del dispositivo, con una estructura estándar definida por el Foro UPnP.

Estos descriptores, tanto de dispositivos como de los servicios, son recuperados por el Punto de Control mediante solicitudes HTTP GET.

Tras esta fase, el Punto de Control ya dispone de la información necesaria para poder proceder con cualquiera de los subsiguientes pasos, Control, Eventos, o Presentación. Este es el

motivo por el que en la figura 2.4, estas etapas aparezcan al mismo nivel y justo por encima de la fase de Descripción.

Control

Una vez obtenido el conocimiento suficiente sobre el dispositivo y sus servicios, un Punto de Control puede invocar acciones de entre las que se definen en los servicios. Para ello, envía mensajes de control a la URL de control del servicio correspondiente (proporcionada en la descripción del dispositivo). En respuesta, este servicio genera un mensaje en el que se incluye la respuesta a la invocación de la acción, si procede, o códigos de error.

Los mensajes de control también se codifican en XML utilizando el formato que define el protocolo SOAP descrito en el punto “Protocolos usados por UPnP”.

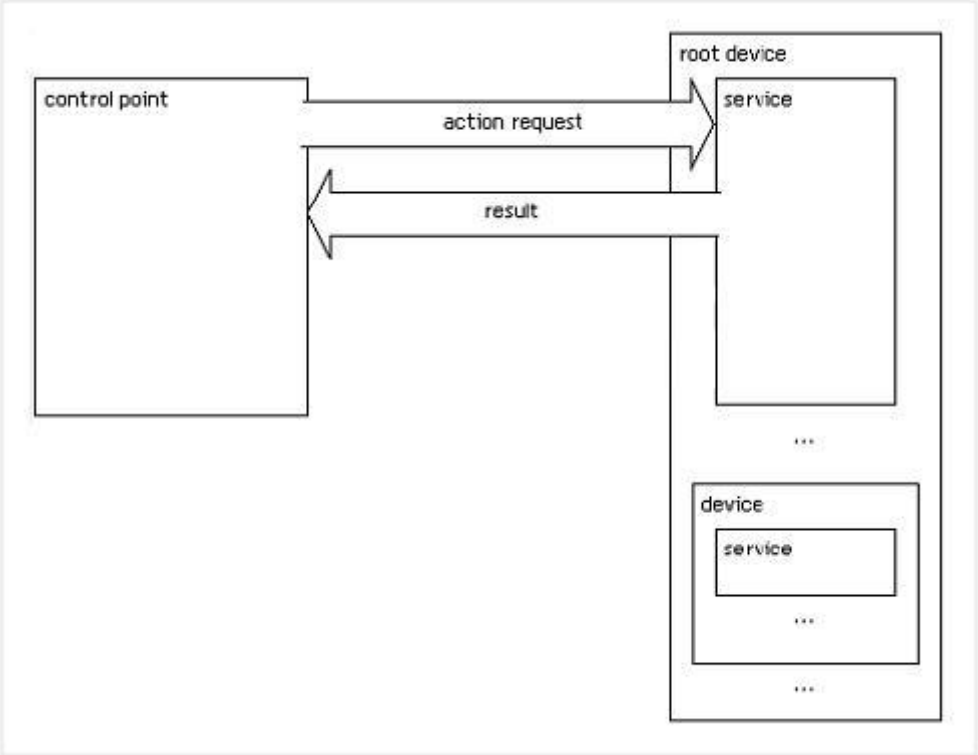


Figura 2.6. Esquema del mecanismo de Control de UPnP [UPNPDEVARCH]

Eventos

Al igual que en el mecanismo anterior, tras disponer el Punto de Control de la información que le facilita las descripciones del dispositivo y sus servicio, ya está en condiciones para proceder con el mecanismo de eventos.

En la etapa de Descripción se dijo que la descripción de un servicio incluye una lista de variables de estado. Algunas de estas variables sufren modificaciones por cambios durante el desarrollo del programa del propio servicio, o bien como consecuencia de la invocación de alguna de las acciones de control definidas para éste.

Si una o más de estas variables generan eventos cuando son modificadas, entonces, el dispositivo debe publicar dichas actualizaciones. Para que un Punto de Control pueda escuchar estos eventos debe haberse subscrito previamente utilizando la URL que se le facilita a tal efecto en la descripción del servicio. De esta forma, el Punto de Control, estaría enterado de cualquier modificación permitiéndole generar un modelo del estado de este servicio en tiempo de ejecución.

Los mensajes de eventos, enviados por los servicios a sus subscriptores, contienen de una a más variables de estado y su valor actual. También estos mensajes se expresan en XML, y se formatean utilizando la arquitectura GENA.

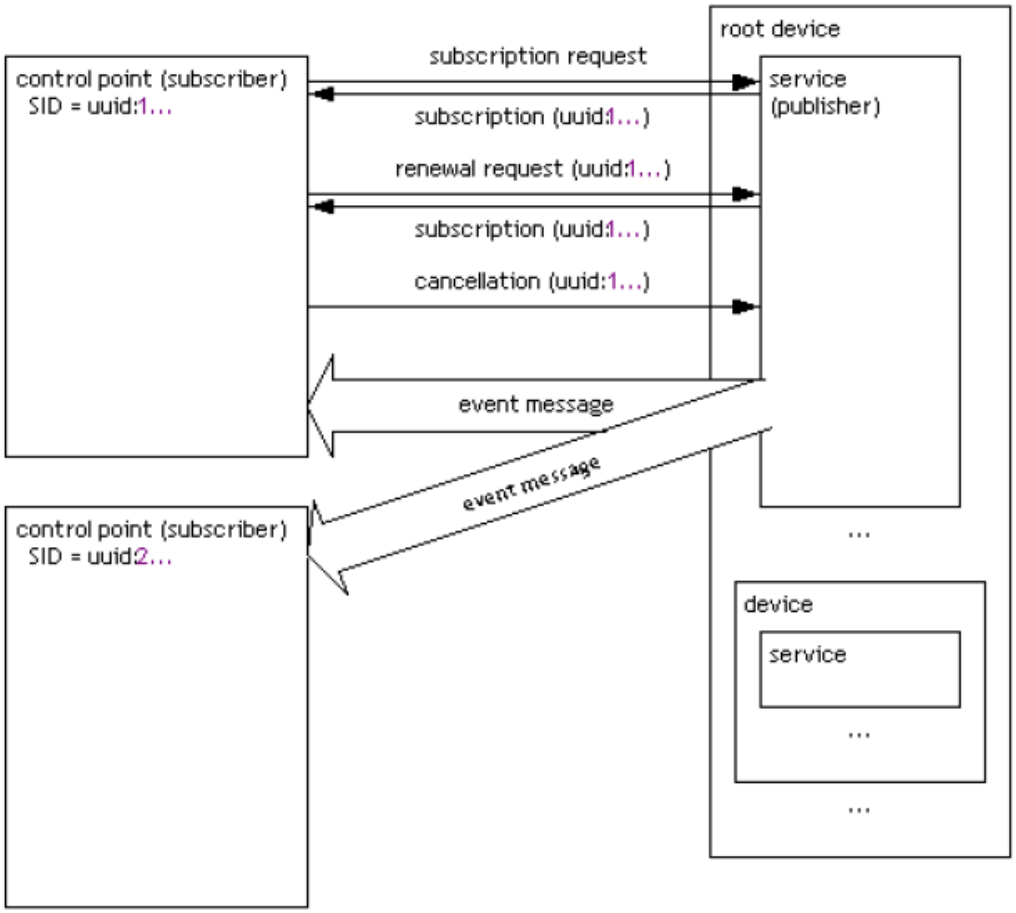


Figura 2.7. Esquema del mecanismo de Eventos de UPnP [UPNPDEVARCH]

Presentación

Alcanzado el mismo nivel de conocimiento de las capacidades del dispositivo que en los dos mecanismos anteriores, el Punto de Control puede recuperar una página en formato *HyperText Markup Language* (HTML) a partir de la URL de presentación que se incluye en la descripción del servicio correspondiente. Esta página es obtenida mediante una solicitud HTTP GET. Se carga en un explorador web y, en función de las capacidades de la misma, permite al usuario controlar al dispositivo y/o visualizar información acerca de su estado actual.

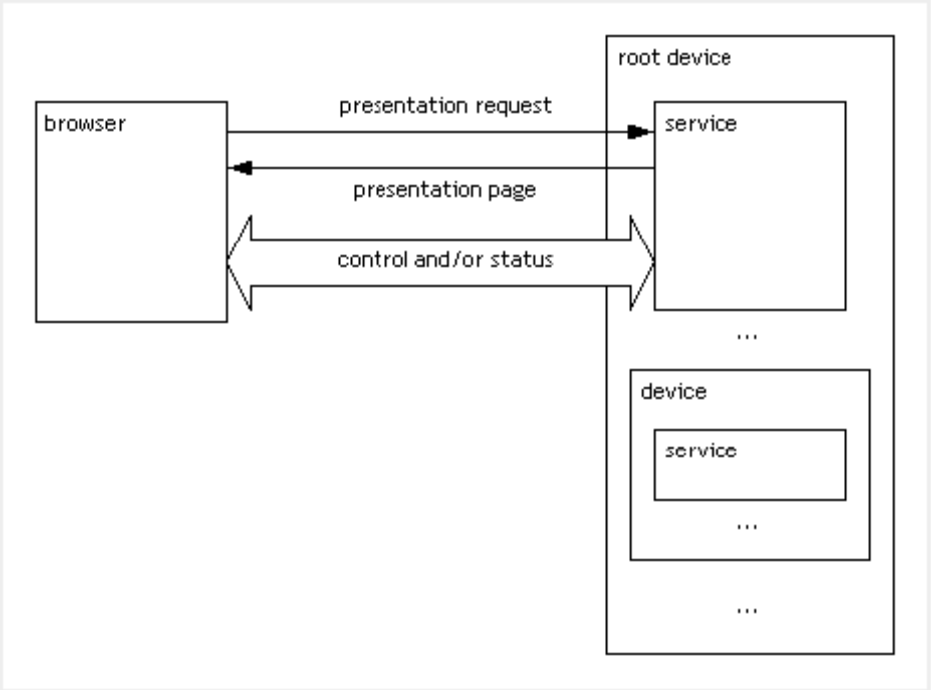


Figura 2.8. Esquema del mecanismo de Presentación de UPnP [UPNPDEVARCH]

2.1.5 Arquitectura UPnP para Audio y Video

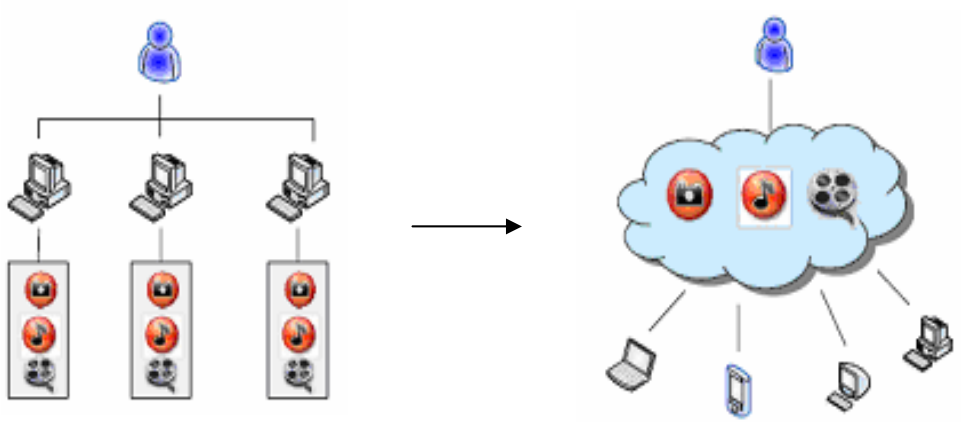
Una vez se han descrito las principales características de UPnP, los protocolos sobre los que se construye y las distintas fases de la comunicación, se dispone del conocimiento necesario para poder presentar la especificación de esta tecnología para escenarios de entretenimiento digital, la arquitectura UPnP de Audio y Video.

La Arquitectura AV UPnP, en inglés *UPnP AV Architecture*, define la interacción entre puntos de control UPnP y dispositivos UPnP que manejan contenidos audiovisuales. [UPNPAV]

Las principales características de esta arquitectura son:

- La independencia del tipo de dispositivo que se use, del formato de contenido y del protocolo de transferencia.
- Habilita la transmisión del contenido de audio y video directamente entre dispositivos sin intervención del Punto de Control.
- La escalabilidad, lo que permite dar soporte tanto a dispositivos con recursos limitados de procesamiento y memoria, como a dispositivos multimedia avanzados.

La principal ventaja de esta arquitectura, como se representa en la Figura 2.9, es la de crear un escenario en el que el contenido multimedia (audio, video e imágenes) que pueda estar almacenado en cualquier dispositivo en red, se comparta y esté disponible en toda la red doméstica para que cualquier dispositivo con capacidad de manejarlo lo haga sin tener que saber la procedencia del mismo.



Escenario No-UPnP Escenario UPnP

Figura 2.9. Escenarios AV en la red doméstica [UPNPMUSE]

Como se ha visto anteriormente, un punto de control controla las operaciones de uno o más dispositivos. Sin embargo, los dispositivos no interaccionan entre sí, de tal forma que toda la coordinación entre ellos es realizada por el Punto de Control. En la arquitectura para Audio y Video de UPnP esto cambia.

Un Punto de Control AV (en adelante *Control Point*) debe interaccionar con dos o más dispositivos que actúan como fuente y sumidero respectivamente. Aunque coordine y sincronice el comportamiento de ambos, los dispositivos interaccionan entre sí para transferir el contenido multimedia desde el primero al segundo mediante un protocolo de transferencia fuera de banda. En esta comunicación entre dispositivos el *Control Point* se mantiene al margen.

Por tanto, en la Arquitectura AV de UPnP se ven involucradas tres tipos de entidades:

- El *Control Point* (Punto de Control).
- El dispositivo que sirve el contenido multimedia, denominado *MediaServer*.
- El dispositivo que consume el contenido multimedia, denominado *MediaRenderer*.

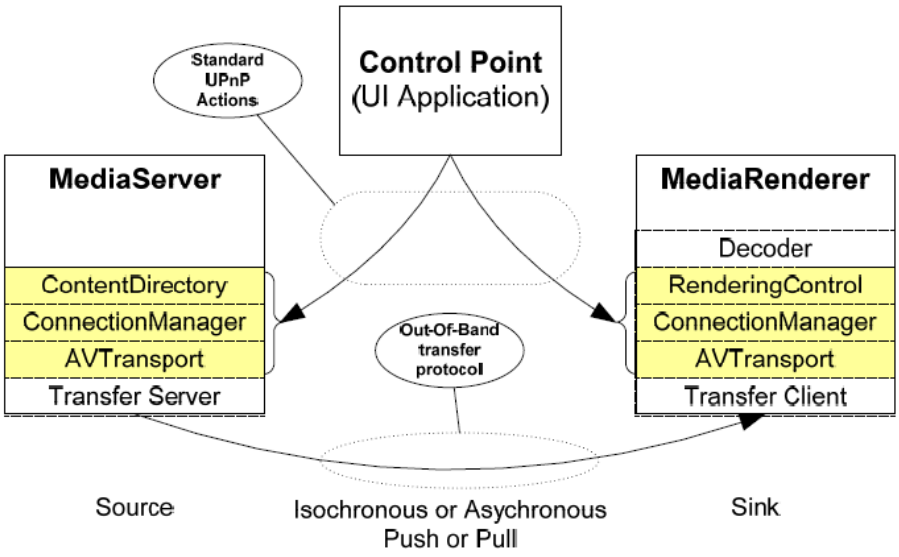


Figura 2.10. Arquitectura AV UPnP [UPNPAV]

AV Control Point

Como se deduce de todo lo dicho anteriormente, un *AV Control Point* coordina las operaciones entra las otras dos entidades, normalmente en respuesta a la interacción del usuario con la interfaz gráfica de éste. La Figura 2.10 muestra esta relación entre las tres entidades.

UPnP MediaServer

Un *MediaServer* es cualquier dispositivo UPnP capaz de localizar el contenido multimedia que está disponible para ser compartido en la red doméstica. Su principal propósito es el de permitir que el *Control Point* pueda listar este contenido para que el usuario pueda seleccionar el que desea renderizar.

Como se puede ver en la Figura 2.10, contiene tres servicios principales:

Content Directory Service:

Permite al *Control Point* enumerar el contenido multimedia que puede ser ofrecido a la red, y obtener detallada información (meta-datos) de cada ítem de este contenido, como por ejemplo el nombre, artista duración, etc. Adicionalmente, esta información puede identificar el protocolo de transferencia y el formato de datos que son soportados por el *MediaServer* para un ítem en concreto. El *Control Point* la utiliza para determinar si un determinado *MediaRenderer* es capaz de reproducir ese ítem.

Connection Manager Service:

Permite al *Control Point* manejar las conexiones asociadas al dispositivo. La principal acción que define es `PrepareForConnection()`. Tiene carácter opcional y es invocada por el *Control Point* para dar la oportunidad al dispositivo de prepararse para la transferencia del contenido. Cuando es implementada, esta acción habilita al *MediaServer* a soportar la transferencia de varios contenidos al mismo tiempo. Para ello genera un identificador de cada conexión, *ConnectioID*, que puede ser usado por otro *Control Point* para averiguar qué conexiones tiene activas el *MediaServer*.

Así también, dependiendo del protocolo de transferencia y el formato del contenido, su invocación puede devolver un *InstanceID* de un *AVTransport Service* que el *Control Point* puede utilizar para controlar el flujo de este contenido (controles comunes como *Play*, *Pause*,…). Ya que un mismo *MediaServer* podría soportar la transferencia de varios contenidos, este identificador pude ser usado para distinguir varias instancias virtuales del servicio *AVTransport*, cada una de las cuales está asociada con la conexión a un *MediaRenderer* distinto.

Si el *Control Point* quiere cerrar una conexión entre *MediaServer* y *MediaRenderer* puede invocar a la acción `ConnectionComplete()` y así liberarla. Esta acción también es opcional.

Si el servicio no implementa la acción `PrepareForConnection()` el *MediaServer* únicamente soportará la conexión con un *MediaRenderer*.

AVTrasnport Service:

Permite al *Control Point* controlar la reproducción del contenido que está asociado con este servicio. Esto incluye los controles comunes como *Play*, *Pause*, *Stop*, etc. Acorde a lo dicho anteriormente, dependiendo del protocolo de transferencia y formatos de contenido que soporte el *MediaServer*, se necesitará o no implementar este servicio. En el caso de que sea necesario, se podrán generar instancias (virtuales) del servicio usando el identificador *InstanceID* y cuyo uso ya se ha comentado en la descripción del anterior servicio.

Si el *MediaServer* no soporta más de una conexión (no implementa la acción `PrepareForConnection()` del *ConnectionManager Service*), los identificadores asociados a la conexión quedan: *ConnectionID* = *AVTransportID* = *RcsID* = 0.

UPnP MediaRenderer

El *MediaRenderer* es el dispositivo UPnP capaz de reproducir y presentar el contenido multimedia que esta disponible en la red doméstica. Su principal característica es que permite al *Control Point* controlar su renderización (Volumen, Mute, Brillo,…) y, más específicamente, podría controlar la reproducción mediante acciones comunes como *Play*, *Pause*, *Stop*,..., siempre y cuando el protocolo de transferencia lo permita.

También en este caso, contiene tres servicios principales (ver Figura 2.10).

Rendering Control Service:

Permite al *Control Point* controlar cómo se renderiza el contenido. Esto incluye aspectos como el brillo, contraste, volumen, mute, etc. Este servicio soporta múltiples y dinámicas instancias para poder mezclar uno o más ítems, como podría ser un mezclador de audio o una ventana "*Picture-in-Picture*". Estas instancias, asociadas a un *InstanceID*, son creadas en la invocación de la acción `PreparaForConnection()` del *Connection Manager Service*.

Connection Manager Service:

Al igual que en el caso del *MediaServer*, este servicio es usado para manejar las conexiones del dispositivo con otro. En el caso del *MediaRenderer*, la acción principal es `GetProtocolInfo()`, la cual permite al *Control Point* enumerar los protocolos de transferencia y los formatos de datos que soporta el *MediaRenderer*, que usará para averiguar si podrá o no reproducir un determinado contenido.

También se implementa la acción `PrepareForConnection()` con la misma utilidad que en el caso del *MediaServer*, incluyendo la generación de identificadores *ConnectionID* para que otro *Control Point* pueda conocer las conexiones activas que tiene el dispositivo.

De igual modo, dependiendo del protocolo de transferencia y el formato del contenido, esta acción puede devolver un *InstanceID* para poder identificar distintas instancias virtuales del servicio *AVTransport Service* que permitan al *Control Point* controlar varios flujos y su reproducción.

Por ultimo, a diferencia del *MediaServer*, también la invocación de esta acción puede generar un *InstanceID* del servicio *RenderingControl Service*. Como se dijo en la descripción de este servicio, estos identificadores son usados por el *Control Point* para el control de la renderización del contenido asociado a cada uno.

Para cerrar una conexión abierta entre el *MediaServer* y el *MediaRenderer* el *Control Point* invoca, si está implementada, la acción `ConnectionComplete()`.

Si el servicio no implementa la acción `PrepareForConnection()` el *MediaRenderer* únicamente soportará la conexión con un *MediaServer*.

AVTransport Service:

Su funcionalidad es la misma que la que se define para el caso del *MediaServer*. Esta es, la del control del flujo y la reproducción. Esto incluye las acciones más comunes, *Play*, *Pause*,… También en el caso del *MediaRenderer* la necesidad de su implementación está condicionada al tipo de protocolo de transferencia que se use. En el caso de que se implemente se podrían generar varias instancias del servicio generando sus respectivos identificadores, *InstanceID*, tal y como se indica en la descripción del anterior servicio.

Se profundizará en las especificaciones de estos servicios asociados a un *UPnP MediaRenderer* en el siguiente apartado.

Sobre la implementación del AVTransport Service:

Como se ha dicho, la implementación del *AVTransport Service* es opcional., tanto en el *MediaServer* como en el *MediaRenderer*. Depende principalmente del protocolo de transferencia del contenido que se use.

Existe una clasificación básica de los protocolos en función de cómo son entregados los datos:

- Pull: la transferencia de datos es iniciada y controlado desde el lado del cliente. HTTP, para contenido en remoto, y FILE, para contenido en local, son ejemplos de protocolo *Pull*.
- Push: la transferencia es iniciada y controlado desde el lado del servidor. RTP es un ejemplo de protocolo *Push*.

En este sentido, un protocolo del tipo *Push* implicaría su implementación en el lado del *MediaServer*. Mientras que un protocolo del tipo *Push* implicaría su implementación en el lado del *MediaRenderer*.

Sobre los identificadores de los servicios:

Si un *MediaServer* o un *MediaRenderer* no soporta más de una conexión (no implementa la acción `PrepareForConnection()` del *ConnectionManager Service*), el identificador de la conexión tomará el valor: *ConnectionID* = 0. Consecuentemente sólo se creará una única instancia de cada servicio *AVTranspor Service* y *RenderingControl Service*, cuyos identificadores tomarán los siguiente valores: *AVTransportID* = *RcsID* = 0.

2.1.6 UPnP MediaRenderer

Volviendo a la definición del apartado anterior, un *UPnP MediaRenderer*, es el dispositivo de la arquitectura AV UPnP que permite la renderización del contenido multimedia compartido en la red de un hogar.

El Foro UPnP especifica los tres tipos de servicios asociados a un *MediaRenderer* presentados con anterioridad. Estos servicios contienen un conjunto de acciones y variables de estado que serán utilizados por un *UPnP AV Control Point* para controlar el proceso de renderización y mantenerse informado sobre el estado de cada uno de ellos.

Puesto que este proyecto trata sobre la implementación de un prototipo de *UPnP MediaRenderer*, y dada su importancia, a continuación se profundizará en los servicios asociados a este tipo de dispositivos.

Teoría de operaciones

El proceso para que un *UPnP MediaRenderer* pueda renderizar el contenido multimedia que dispone los *UPnP MediaServer*s comienza con la ya descrita fase de descubrimiento, en la que el dispositivo es descubierto por el *Control Point*.

Tras recuperar el descriptor del dispositivo y el de sus servicios asociados, el *Control Point* ya dispone de la información necesaria para subscribirse al mecanismo de eventos de los servicios e invocar las acciones definidas en estos.

Mediante la invocación de acciones del *ConnectionManager Service* del dispositivo, el *Control Point* prepara la transferencia del contenido.

Tras esto, el usuario por medio del *Control Point* puede invocar las acciones que le permiten iniciar la reproducción y controlarla definidas en el *AVTrasnsport Service*.

De la misma forma, puede invocar acciones del *RenderingControl Service* para controlar cómo se renderiza el contenido (volumen, brillo,…).

El algoritmo que se describe a continuación define de forma general los posibles procesos de comunicación entre las tres entidades que intervienen (*Control Point*, *MediaServer* y *MediaRenderer*).

1. Descubrimiento, usando el mecanismo UPnP de Descubrimiento, de los *MediaServer*s y *MediaRenderer*s conectados a la red doméstica.

2. Localización del contenido que se desea reproducir por medio del servicio *ContentDirectory Service* del *MediaServer*, invocando la acción `Browse()` o `Search()`. Entre la información devuelta de la llamada a cualquiera de ellos figura los protocolos de transferencia y los formatos de los datos que el *MediaServer* soporta.

3. Obtención de los formatos y protocolos soportados por el *MediaRenderer* invocando el método `GetProtocolInfo()` el servicio *ConnectioManager* Service.
4. Comparación de los formatos y protocolos entre *MediaRenderer* y *MediaServer*. El *Control Point* selecciona un protocolo de transferencia y un formato de datos que sean soportados por ambos.

5. Preparación de los dispositivos para la comunicación utilizando la llamada a la acción `PrepareForConnection()`, si está implementada, de los respectivos servicios *ConnectionManager Service*, obteniendo las instancias necesarias para el manejo de las conexiones, tal y como se indica en las anteriores descripciones de este servicio.

6. Selección del ítem deseado de entre los ítems que se listan disponibles en el *Control Point*. Se utiliza la llamada a la acción `SetAVTrasnportURI()` del servicio *AVTransport Service* para identificar a este contenido.

7. Iniciación de la transferencia. Se invoca la acción `Play()` del servicio *AVTransport Service* que inicia el proceso de reproducción. Se pueden invocar otras acciones del servicio como `Stop()`, `Pause()`, `Seek()`,... para el control sobre la reproducción y el flujo.

8. Ajuste de características de la renderización: volumen, mute,… por medio de la llamada a acciones del servicio *RenderingControl Service*, tales como, `SetVolume()`, `SetMute()`, etc.

9. Selección el siguiente contenido de forma análoga al paso 6 o bien por medio del mecanismo que se proporciona en la acción `SetNextAVTransportURI()`.

10. Cierre de la conexión, cuando la sesión sea finalizada, mediante la invocación, si está implementada, de la acción `ConnectionComplete()` de los respectivos servicios *ConnectionManager Service* del *MediaServer* y *MediaRenderer*.

En la siguiente figura se muestra un esquema de este algoritmo.

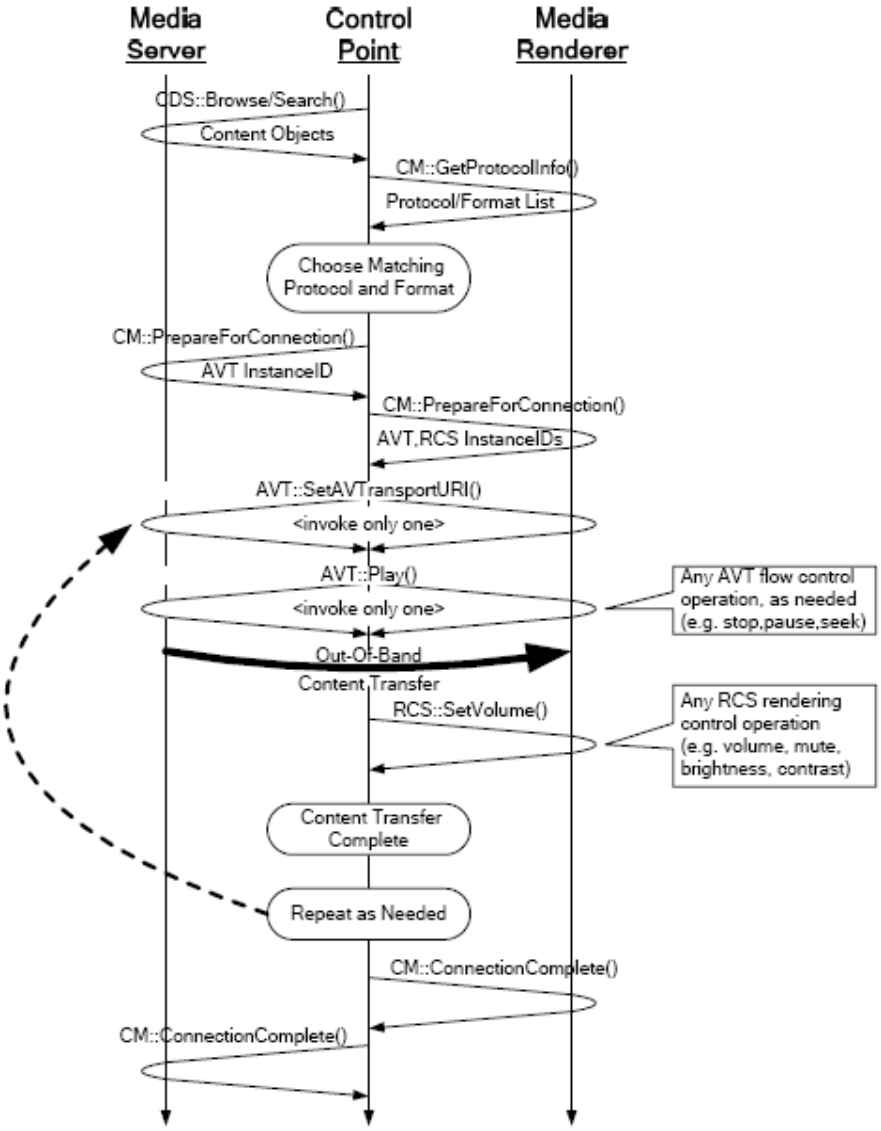


Figura 2.11. Diagrama general de interacción [UPNPAV]

El significado de las siglas que se utilizan se corresponde con los servicios anteriormente descritos:

CDS: *ContentDirectory Service* (del *MediaServer*)
CM: *ConnectionManager Service* (del *MediaServer* y *MediaRenderer*)
AVT: *AVTransport Service* (del *MediaServer* o el *MediaRenderer*)
RCS: *RenderingControl Service* (del *MediaRenderer*)

Servicios de un *UPnP MediaRenderer*

Un servicio UPnP se caracteriza por contener un conjunto de acciones y un conjunto de variables de estado.

Las acciones son procesos que un *Control Point* puede invocar remotamente sobre el *MediaRenderer*. Tiene asociados unos argumentos de entrada y/o unos argumentos de salida. Cada uno de estos argumentos tiene asociada una variable de estado del servicio.

Las variables de estado permiten modelar en cada momento el estado en el que se encuentra el servicio, y por extensión el dispositivo que lo contiene. El cambio de valor en algunas de estas variables provoca la generación de un evento que es enviado a los subscriptores, generalmente un *Control Point*, para que puedan advertir el cambio en el modelo de estado del servicio. Esto implica que un argumento que modifique el valor de una variable de estado que genera eventos provoque el envío de un evento para informar del cambio.

Los tres servicios definidos para un *UPnP MediaRenderer* son, como ya se dijo, *RenderingControl Service*, *ConnectionManager Service* y *AVTransport Service*.

Entre las variables de estado de estos servicios, existe un conjunto formado por aquellas cuyo nombre comienza por "A_ARG_TYPE_...". Este tipo de variables no provocan el envío de eventos, de hecho no cambian. Esencialmente su función es la de proporcionar información del tipo de datos que contiene el argumento asociado a ella.

Existe una variable de estado especial y definida para los servicios *RenderingControl Service* y *AVTransport Service*. Se trata de la variable `LastChange`. Su propósito es el de recopilar información sobre cambios que se han producido en variables de estado que no generan eventos cuando cambian sus valores. Periódicamente la información sobre estos cambios es enviada al *Control Point* mediante un único evento asociado a esta variable.

El mecanismo de eventos definido para el envío de eventos por cambios producidos en las variables de estado, ya sea directamente o indirectamente a través de la variable `LastChange`, permite al *Control Point* tener permanentemente el modelo del estado en el que se encuentra cada servicio y por extensión del *MediaRenderer*.

RenderingControl Service

Este servicio proporciona al *Control Point* la capacidad de controlar algunos aspectos de la renderización del contenido. Por tanto, le permite modificar características como el brillo, el contraste de la imagen o el volumen del audio entre muchas otras.

Las acciones que define este servicio son:

Acción	**Descripción**
`ListPresets()`	[Obligatoria] Devuelve una lista con los *presets*[1] definidos
`SelectPreset()`	[Obligatoria] Reestablece las variables de estado a los valores definidos por el preset especificado
`GetBrightness()`	[Opcional] Devuelve el valor actual de la variable de estado `Brightness`
`SetBrightness()`	[Opcional] Establece la variable de estado `Brightness` al valor especificado
`GetContrast()`	[Opcional] Devuelve el valor actual de la variable de estado `Contrast`
`SetContrast()`	[Opcional] Establece la variable de estado `Contrast` al valor especificado
`GetSharpness()`	[Opcional] Devuelve el valor actual de la variable de estado `Sharpness`
`SetSharpness`	[Opcional] Establece la variable de estado `Sharpness` al valor especificado
`GetRedVideoGain()`	[Opcional] Devuelve el valor actual de la variable de estado `RedVideoGain`
`SetRedVideoGain()`	[Opcional] Estable la variable de estado `RedVideoGain` al valor especificado
`GetGreenVideoGain()`	[Opcional] Devuelve el valor actual de la variable de estado `GreenVideoGain`

[1] *Preset*: en sistemas audiovisuales un "preset" es un conjunto de parámetros o variables del sistema que se memorizan definiendo una configuración del mismo.

`SetGreenVideoGain()`	[Opcional] Estable la variable de estado `GreenVideoGain` al valor especificado
`GetBlueVideoGain()`	[Opcional] Devuelve el valor actual de la variable de estado `BlueVideoGain`
`SetBlueVideoGain()`	[Opcional] Estable la variable de estado `BlueVideoGain` al valor especificado
`GetRedVideoBlackLevel()`	[Opcional] Devuelve el valor de la variable de estado `RedVideoBlackLevel`
`SetRedVideoBlackLevel`	[Opcional] Establece la variable de estado `RedVideoBlackLevel`
`GetGreenVideoBlackLevel()`	[Opcional] Devuelve el valor de la variable de estado `GreenVideoBlackLevel`
`SetGreenVideoBlackLevel`	[Opcional] Establece la variable de estado `GreenVideoBlackLevel`
`GetBlueVideoBlackLevel()`	[Opcional] Devuelve el valor de la variable de estado `BlueVideoBlackLevel`
`SetBlueVideoBlackLevel`	[Opcional] Establece la variable de estado `BlueVideoBlackLevel`
`GetColorTemperature()`	[Opcional] Devuelve el valor actual de la variable de estado `ColorTemperature`
`SetColorTemperature()`	[Opcional] Establece el valor de la variable de estado `ColorTemperature`
`GetHorizontalKeystone()`	[Opcional] Devuelve el valor actual de la variable de estado `HorizontalKeystone`
`SetHorizontalKeystone()`	[Opcional] Establece la variable de estado `HorizontalKeystone` al valor especificado
`GetVerticalKeystone()`	[Opcional] Devuelve el valor actual de la variable de estado `VerticalKeystone`
`SetVerticalKeystone()`	[Opcional] Establece la variable de estado `VerticalKeystone` al valor especificado
`GetMute()`	[Opcional] Devuelve el valor actual de la variable de estado `Mute` para un determinado canal
`SetMute()`	[Opcional] Establece la variable de estado *Mute* al valor especificado para un canal específico
`GetVolume()`	[Opcional] Devuelve el valor actual de la variable de estado `Volume` para un determinado canal
`SetVolume()`	[Opcional] Establece la variable de estado `Volume` al valor especificado para un canal específico
`GetVolumeDB()`	[Opcional] Devuelve el valor actual de la variable de estado `VolumeDB` para un determinado canal
`SetVolumeDB()`	[Opcional] Establece la variable de estado `VolumeDB` al valor especificado para un canal concreto
`GetVolumeDBRange()`	[Opcional] Devuelve el rango válido, determinado por sus valores máximo y mínimo, para la variable de estado `VolumeDB` para un canal concreto
`GetLoudness()`	[Opcional] Devuelve el valor actual de la variable de estado `Loudness` para un canal determinado
`SetLoudness()`	[Opcional] Establece la variable de estado `Loudness` al valor especificado para un canal concreto
`GetStateVariables()`	[Opcional] Devuelve la colección actual de nombres de las variables de estado de este servicio y sus respectivos valores
`SetStateVariables()`	[Opcional] Extrae los valores del argumento de entrada `StateVariableValuePairs` y la copia a las correspondientes variables de estado asociadas. Los argumentos `RenderingControlUDN`, `ServiceType` y `ServiceID` se usan para comprobar la compatibilidad

Tabla 2.1. Acciones del servicio *RenderinControl Service*

A continuación se muestra una tabla con las variables de estado del servicio *RenderingControl Service* y una breve descripción:

Variable de Estado	**Ob/Op**	**Evento**	**Descripción**
`LastChange`	Ob	Si	Recopila información sobre cambios que se han producido en variables de estado que no generan eventos cuando cambian sus valores
`PresetNameList`	Ob	No	Contiene una lista de los nombres de presets válidos del dispositivo.
`Brightness`	Op	No	Representa el valor del brillo de la imagen
`Contrast`	Op	No	Representa el valor del contraste de la imagen
`Sharpness`	Op	No	Representa el valor del contraste de la imagen
`RedVideoGain`	Op	No	Representa el valor del control de ganancia de rojos de la pantalla
`GreenVideoGain`	Op	No	Representa el valor del control de ganancia de verdes de la pantalla
`BlueVideoGain`	Op	No	Representa el valor del control de ganancia de azules de la pantalla
`RedVideoBlackLevel`	Op	No	Representa el valor de la intensidad mínima de salida del rojo de la pantalla
`GreenVideoBlackLevel`	Op	No	Representa el valor de la intensidad mínima de salida del verde de la pantalla
`BlueVideoBlackLevel`	Op	No	Representa el valor de la intensidad mínima de salida del azul de la pantalla
`ColorTemperature`	Op	No	Representa el valor de la característica de la pantalla: calidad de color del blanco
`HorizontalKeystone`	Op	No	Representa el valor del nivel de compensación de la distorsión horizontal de la pantalla
`VerticalKeystone`	Op	No	Representa el valor del nivel de compensación de la distorsión vertical de la pantalla
`Mute`	Op	No	Representa el valor de mute del asociado a un canal de audio
`Volume`	Op	No	Representa el valor de volumen del asociado a un canal de audio
`VolumeDB`	Op	No	Representa el valor de volumen en la escala de decibelios del asociado a un canal de audio
`Loudness`	Op	No	Representa el valor que indica si el *Loudness* del sistema de sonido está activado
`A_ARG_TYPE_Channel`	Ob	No	Provee información de tipo para el argumento `Channel` en varias acciones
`A_ARG_TYPE_InstanceID`	Ob	No	Provee información de tipo para el argumento `InstanceID` en varias acciones
`A_ARG_TYPE_PresetName`	Ob	No	Provee información de tipo para el argumento `PresetName` en la acción `SelectPreset()`
`A_ARG_TYPE_DeviceUDN`	Op	No	Provee información de tipo para el argumento `RenderingControlUDN` en ciertas acciones
`A_ARG_TYPE_ServiceType`	Op	No	Provee información de tipo para el argumento `ServiceType`
`A_ARG_TYPE_ServiceID`	Op	No	Provee información de tipo para el argumento `ServiceID` en ciertas acciones

A_ARG_TYPE_ StateVariableValuePairs	Op	No	Provee información de tipo para el argumento StateVariableValuePairs en ciertas acciones
A_ARG_TYPE_ StateVariableList	Op	No	Provee información de tipo para el argumento StateVariableList en ciertas acciones

Tabla 2.2. Variables de estado del servicio *RenderinControl Service*

Ob/Op: implementación: Obligatoria / Opcional
Evento: si un cambio en su valor provoca el envío de un evento: Si / No

ConnectionManager Service

Este servicio permite al dispositivo prepararse para la conexión entrante y al *Control Point* manejar las conexiones asociadas a este.

Las acciones que define este servicio son:

GetProtocolInfo()	[Obligatoria] Devuelve una lista de los protocolos que el servicio *ConnectionManager Service* soporta en su estado actual
PrepareForConnectio()	[Opcional] Permite al dispositivo prepararse para la recepción de los datos multimedia, y también, si puede o no realizar la conexión en base al estado actual y el estado de la red
ConnectionComplete()	[Opcional] Es usada para informar al dispositivo que una conexión, previamente abierta, ya no es necesaria
GetCurrentConnectionIDs()	[Obligatoria] Devuelve una lista de identificadores, ConnectionID, de las conexiones actualmente activas
GetCurrentConnectionInfo()	[Obligatoria] Devuelve información relativa a la conexión a la que hace referencia el argumento ConnectionID

Tabla 2.3. Acciones del servicio *ConnectionManager Service*

A continuación se muestra una tabla con las variables de estado del servicio *ConnectionManager Service* y una breve descripción:

Variable de Estado	**Ob/Op**	**Evento**	**Descripción**
SourceProtocolInfo	Ob	Si	Contiene la lista de protocolos que este servicio soporte para la transmisión del contenido multimedia
SinkProtocolInfo	Ob	Si	Contiene la lista de protocolos que este servicio soporte para la recepción del contenido multimedia
CurrentConnectionIDs	Ob	Si	Contiene una lista de los identificadores de las conexiones activas
A_ARG_TYPE_ ConnectionStatus	Ob	No	Provee información de tipo para el argumento Status en la acción GetCurrentConnectionInfo()
A_ARG_TYPE_ ConnectionManager	Ob	No	Provee información de tipo para el argumento PeerConnectionManager en las acciones GetCurrentConnectionInfo() y PrepareForConnection()
A_ARG_TYPE_Direction	Ob	No	Provee información de tipo para el argumento Direction en la acción PrepareForConnection()
A_ARG_TYPE_ProtocolInfo	Ob	No	Provee información de tipo para los argumentos RemoteProtocolInfo en la acción GetCurrentConnectionInfo() y ProtocolInfo en la acción

			GetCurrentConnectionInfo()
A_ARG_TYPE_ConnectionID	Ob	No	Provee información de tipo para el argumento ConnectionID en las acciones PrepareForConnection(), ConnectionComplete() y GetCurrentConnectionInfo()
A_ARG_TYPE_ AVTrasnportID	Ob	No	Provee información de tipo para el argumento AVTrasnport en las acciones GetCurrentConnectionInfo() y PrepareForConnection()
A_ARG_TYPE_RcsID	Ob	No	Provee información de tipo para el argumento Rcs en las acciones GetCurrentConnectionInfo() y PrepareForConnection()

Tabla 2.4. Variables de estado del servicio *ConnectionManager Service*

AVTransport Service

Este servicio permite al *Control Point* controlar el flujo y la reproducción del contenido multimedia que se está renderizando.

Las acciones que define son las siguientes:

SetAVTansportURI()	[Obligatoria] Especifica la URI del recurso que va a ser controlado a través de este servicio
SetNextAVTrasnportURI()	[Opcional] Especifica la URI del recurso a ser controlado cuando finalice la reproducción en curso
GetMediaInfo()	[Obligatoria] Devuelve información asociada al recurso multimedia que está siendo controlado
GetMediaInfo_Ext()	[Obligatoria] Devuelve la misma información que la anterior acción pero añadiendo la categoría de contenido multimedia
GetTransportInfo()	[Obligatoria] Devuelve información sobre el estado del transporte y reproducción del recurso
GetPositionInfo()	[Obligatoria] Devuelve información sobre el momento de la reproducción en que se encuentra el recurso como puede ser información sobre la pista actual, su referencia temporal, su duración,…
GetDeviceCapabilities()	[Obligatoria] Devuelve información sobre las capacidades de procesamiento del dispositivo
GetTransportSettings()	[Obligatoria] Devuelve información sobre algunos características configurables como el modo de reproducción o la calidad de la grabación
Stop()	[Obligatoria] Para la reproducción del recurso actual
Play()	[Obligatoria] Arranca la reproducción del recurso actual a una velocidad determinada y de acuerdo la forma que indique el modo de reproducción (CurrentPlayMode)
Pause()	[Opcional] Pausa la reproducción en curso, siempre que ésta esté en el estado: *Playing*
Record()	[Opcional] Arranca el proceso de grabación del recurso actual de acuerdo al modo de calidad especificado anteriomente
Seek()	[Obligatoria] Permite la búsqueda dentro del archivo multimedia de una posición concreta especificada en el argumento Target. El argumento Unit define cómo debe ser interpretado el otro argumento.
Next()	[Obligatoria] Permite avanzar a la siguiente pista
Previous()	[Obligatoria] Permite retroceder a la pista anterior
SetPlayMode()	[Opcional] Establece el modo de reproducción

`SetRecordQualityMode()`	[Opcional] Establece el modo de calidad de la grabación
`GetCurrentTransportActions()`	[Opcional] Devuelve la variable de estado `CurrentTransportActions`
`GetDRMState()`	[Opcional] Devuelve información sobre el estado actual del DRM
`GetStateVariables()`	[Opcional] Devuelve la colección actual de nombres de las variables de estado de este servicio y sus respectivos valores
`SetStateVariables()`	[Opcional] Extrae los valores del argumento de entrada `StateVariableValuePairs` y la copia a las correspondientes variables de estado asociadas. Los argumentos `AVTransportUDN`, `ServiceType` y `ServiceID` se usan para comprobar la compatibilidad

Tabla 2.5. Acciones del servicio *AVTransport Service*

La siguiente tabla muestra las variables de estado del servicio *AVTransport Service* y una breve descripción de las mismas:

Variable de Estado	**Ob/Op**	**Evento**	**Descripción**
`TransportState`	Ob	No	Principal variable de estado del servicio. Define los estados del transporte (o la renderización).
`TransportStatus`	Ob	No	Sirve para informar al *Control Point* de posibles fallos durante el control del recurso, como puede ser fallos con la red
`CurrentMediaCategory`	Ob	No	Indica si el recurso actual es *track-aware* o *track-unaware*[1]
`PlaybackStorageMedium`	Ob	No	Indica el tipo de medio donde está almacenado el recurso especificado
`RecordStorageMedium`	Ob	No	Indica el tipo de medio donde almacenar el recurso en el proceso de grabación
`PossiblePlaybackStorageMedia`	Ob	No	Contiene una lista de los posibles medios de almacenamiento del contenido multimedia soportados por el dispositivo para su reproducción
`PossibleRecordStorageMedia`	Ob	No	Contiene una lista de los posibles medios donde almacenar el contenido multimedia soportados por el dispositivo para la grabación
`CurrentPlayMode`	Ob	No	Indica el modo actual de reproducción (Por ejemplo, “aleatoria”, “repeticion”, etc)
`TransportPlaySpeed`	Ob	No	Indica la velocidad, relativa a la normal, de reproducción
`RecordMediumWriteStatus`	Ob	No	Indica el tipo de protección sobre escritura que tiene el medio donde ha sido descargado el recurso
`CurrentRecordQualityMode`	Ob	No	Indica el modo de calidad actual con la que se graba el recurso
`PossibleRecordQualityModes`	Ob	No	Contiene una lista de los posibles modos de calidad con la que se puede grabar un recurso
`NumberOfTracks`	Ob	No	Contiene el número de pistas[1] controladas por este servicio

[1] *Track-aware/unaware*: se clasifica como *track-aware* al recurso compuesto por varias pistas (5) definidas e independientes (multiple) (ejemplo: CD), o una única (*single*) (ejemplo un fichero de audio mp3). Se entiende por recurso *track-unaware* aquel que no identifica pistas distintas por lo que en sí mismo conforma una única unidad (ejemplo: VHS).

CurrentTrack	Ob	No	Contiene el numero de secuencia que identifica a la pista que está seleccionada
CurrentTrackDuration	Ob	No	Contiene la duración de la pista actual
CurrentMediaDuration	Ob	No	Contiene la duración del recurso multimedia, la suma de duraciones de cada una de sus pistas, identificado por el AVTransportURI
CurrentTrackMetaData	Ob	No	Contiene los meta-datos[2] asociados a la pista a la que hace referencia la variable de estado CurrentTrackURI
CurrentTrackURI	Ob	No	Contiene una referencia, en formato URI[3] , de la pista actual
AVTransportURI	Ob	No	Contiene una referencia, en formato URI, del recurso controlado por este servicio
AVTransportURIMetaData	Ob	No	Contiene los meta-datos asociados al recuso al que hace referencia la variable de estado AVTransportURI
NextAVTransportURI	Ob	No	Contiene el valor de la variable AVTransportURI que para el próximo recurso a controlar
NextAVTransportURIMetaData	Ob	No	Contiene los meta-datos asociados al recuso al que hace referencia la variable de NextAVTransportURI
RelativeTimePosition	Ob	No	Contiene la posición actual dentro de la pista, en términos de tiempo, medido desde el principio de ésta (para *track-aware*)
AbsoluteTimePosition	Ob	No	Contiene la posición actual, en terminos de tiempo, relativa al recurso completo y medida desde el principio de éste
RelativeCounterPosition	Ob	No	Contiene la posición actual dentro de la pista, en términos de un contador sin dimensión, medido desde el principio de ésta (para *track-aware*)
AbsoluteCounterPosition	Ob	No	Contiene la posición actual, en términos de un contador sin dimensión, relativa al recurso completo y medida desde el principio de éste
CurrentTransportActions	Op	No	Contiene una lista de las acciones de control del transporte y la reproducción que pueden se invocadas para el recurso actual (Por ejemplo, Play(), Pause(), etc)
LastChange	Ob	Si	Recopila información sobre cambios que se han producido en variables de estado que no generan eventos cuando cambian sus valores
DRMState	Op	No	Usada por las implementaciones del

[1] Pista: es cada uno de los ítems o piezas diferentes de que se compone un recurso multimedia. Por ejemplo un CD de música es un recurso multimedia formado generalmente por 17 canciones aproximadamente, cada una de las cuales conforman una pista (*track-aware*). Un recurso completo puede consistir en una única pista, como por ejemplo, un fichero almacenado en disco que contiene una única canción o que aún conteniendo varias, éstas conforman una única unidad (*track-unaware*).

[2] Meta-datos: información asociada al contenido multimedia, como puede ser el título, el autor, la duración, el tipo de contenido,…, representada en formato XML.

[3] URI (*Uniform Resource Identifie*r): es una cadena corta de caracteres que identifica de forma unívoca un recurso.

			servicio que manejan contenidos con control DRM[1]
A_ARG_TYPE_SeekMode	Ob	No	Provee información de tipo para el argumento `Unit` en la acción `Seek()`
A_ARG_TYPE_SeekTarget	Ob	No	Provee información de tipo para el argumento `Target` en la acción `Seek()`
A_ARG_TYPE_InstanceID	Ob	No	Provee información de tipo para el argumento `InstanceID` en varias acciones
A_ARG_TYPE_DeviceUDN	Op	No	Provee información de tipo para el argumento `RenderingControlUDN` en ciertas acciones
A_ARG_TYPE_ServiceType	Op	No	Provee información de tipo para el argumento `ServiceType`
A_ARG_TYPE_ServiceID	Op	No	Provee información de tipo para el argumento `ServiceID` en ciertas acciones
A_ARG_TYPE_ StateVariableValuePairs	Op	No	Provee información de tipo para el argumento `StateVariableValuePairs` en ciertas acciones
A_ARG_TYPE_ StateVariableList	Op	No	Provee información de tipo para el argumento `StateVariableList` en ciertas acciones

Tabla 2.6. Variables de estado del servicio *AVTransport Service*

Estados del Transporte (renderización)

De entre estas variables hay que destacar por su importancia en la implementación de un *MediaRenderer*, la variable *TransportState*. Como se ha dicho, define los estados en los que se encuentra el transporte del contenido, o interpretado de otra manera, los estados de la renderización. Estos son:

- *NO_MEDIA_PRESENT* → Sin contenido asociado.
- *STOPPED* → Parado.
- *TRANSITIONING* → En proceso de transición.
- *PLAYING* → Renderizando el medio.
- *PAUSED_PLAYBACK* → Renderización pausada.
- *RECORDING* → Grabando el medio.
- *PAUSED_RECORDING* → Grabación pausada.

La principal consideración a tener en cuenta es que si surge un error en cualquier estado de la renderización, se debe volver al estado *STOPPED*.

2.1.7 Limitaciones de UPnP

Como toda tecnología, UPnP asume ciertas limitaciones. A continuación se detallan las más destacadas:

- Seguridad: UPnP no dispone por defecto de un mecanismo de autenticación. Sin embargo, sí dispone de DCPs, descripciones de servicios, que incorporan políticas de seguridad a la arquitectura UPnP. El servicio de seguridad para dispositivos de UPnP (*UPnP DeviceSecurity Service*) proporciona los servicios

[1] DRM (*Digital Rights Management*): mecanismo de protección del contenido para la gestión de derechos digitales.

necesarios para la autenticación, autorización, prevención de repeticiones y privacidad para las acciones SOAP de UPnP. [UPNPSECURITY]

- QoS: tampoco UPnP por defecto incorpora políticas que aseguren un buen servicio acorde al tipo de datos, cantidad y tiempo necesario para ser transmitidos, especialmente indicadas en transmisiones de voz y video. Aunque, como en el caso anterior, si provee de DCPs que definen una arquitectura para la gestión de políticas de calidad de servicio. [UPNPQOS]

- En cuanto a la Arquitectura AV de UPnP [UPNPAV]:
  - No permite comunicaciones interactivas (Ejemplo: videoconferencia)
  - No provee de ningún sistema de protección de contenido, tipo DRM, para la gestión de derechos digitales.
  - No proporciona soporte para la reproducción sincronizada de un medio a múltiples dispositivos de renderización[1].

2.1.8 Justificación del uso de UPnP

La realización de este proyecto viene motivada por la implementación de un dispositivo conforme a la especificación UPnP para *MediaRenderer*s. Define un conjunto de protocolos para facilitar la comunicación entre entidades conectadas en una misma red sin que el usuario tenga que realizar complicadas configuraciones ni instalar drivers para su uso. Por este hecho, resulta una tecnología realmente interesante para su aplicación en sistemas de entretenimiento en los que participan varias entidades que permiten compartir el contenido multimedia en la red y el manejo remoto de su renderización.

2.2 JMF

En este apartado se presenta la librería Java para aplicaciones multimedia, *Java Media Framework* (JMF). Se describirán las principales clases e interfaces del API que define, con especial atención a la clase `Player`, así como la especificación que define para el *streaming* RTP. Seguidamente se enumerarán las distintas opciones para extender las capacidades de la misma. Y por último, se justificará su uso en este proyecto.

2.2.1 Introducción

Java Media Framework[2] (JMF) es una librería Java que extiende a la *Java 2 Platform*, *Standard Edition* (J2SE), proporcionando un entorno de trabajo Java para el desarrollo de aplicaciones Java y Applets que manejen medios basados en tiempo (*time-based media*). Los medios basados en tiempo son medios tales como el audio, el video, MIDI y animaciones que cambian con el tiempo. [JMF]

Características principales de JMF en su última versión:

- Programación Multiplataforma: al estar implementado en Java asume ésta característica del lenguaje.
- Estabilidad: proporciona estabilidad al funcionar sobre la Máquina Virtual Java (JVM).
- Sencillez: permite realizar tareas multimedia complejas usando unos pocos comandos.
- Manipulación integral de medios: permite la captación, procesamiento, reproducción, almacenamiento y difusión de medios.

[1] Renderización: en el contexto de este proyecto, se refiere a la última fase del procesado multimedia en la que los datos de audio, video o imagen son preparados para poder ser enviados a los respectivos sistemas de sonido (altavoces) y visualización (pantalla).
Renderizar: acción de realizar la renderización.

[2] *Framework*: conjunto de librerías software cuyo código provee funcionalidad básica, pudiendo ser sobrescrito y especializado por el código del programador. Proporciona un entorno de desarrollo de código.

- Flexibilidad: permite a desarrolladores y proveedores de tecnología la implementación de soluciones personalizadas en base al API existente e integrar fácilmente nuevas tecnologías en el *Framework*.
- Extensibilidad: permite el desarrollo de *plug-ins* JMF. Estos son, demultiplexadores, *codecs* (codificadores y decodificadores), procesadores de efectos, multiplexadores y renderizadores.

En la siguiente figura se muestra la arquitectura software de JMF. Como se puede ver, está basada en una Máquina Virtual Java. Los *plug-ins* se definen como módulos que son registrados por el *Framewok* para poder ser utilizados.

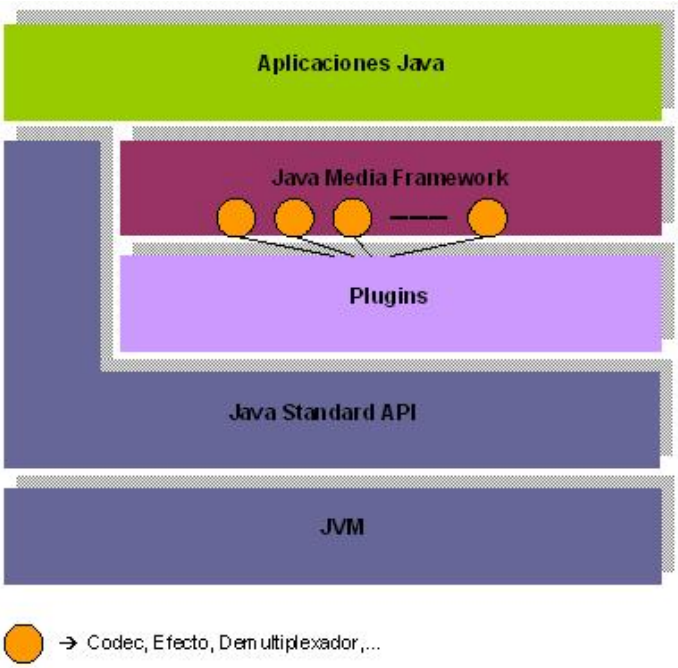


Figura 2.12. Capas de la arquitectura software de JMF

La Figura 2.13 representa una analogía del esquema de funcionamiento de JMF, similar al proceso de reproducción de un DVD. Una fuente de datos encapsula el media stream, flujo de medios, multiplexado (audio y video), como lo hace un disco DVD. Posteriormente un reproductor DVD, proporciona el procesamiento, principalmente la decodificación, y los mecanismos de control. Por último, este reproductor demultiplexa la señal en una señal de audio y otra señal de video, y las proporciona a los correspondientes renderizadores, como puede ser una pantalla y unos altavoces. También, puede transferir la señal procesada para su almacenaje en algún medio de almacenamiento (cinta, CD, DVD, memoria USB,…). O bien, puede ser enviada a través de la red utilizando un protocolo de transferencia como RTP (*Real-time Transport Protocol*). [JMFGUIA]

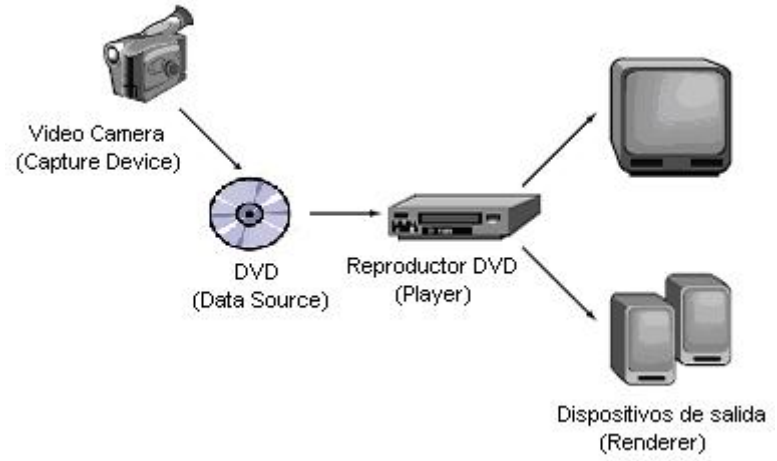


Figura 2.13. Analogía del funcionamiento de JMF [JMFGUIA]

Un flujo de medios es el conjunto de datos obtenidos de un fichero (multimedia) local, o adquiridos de la red, o captados por una cámara o un micrófono. Los flujos de medios suelen contener múltiples canales de datos llamados pistas. El tipo de pista identifica la clase de datos que contiene, como audio o video y el formato de la pista define cómo están estructurados sus datos. El conjunto de ambas informaciones define el tipo de contenido (*Content Type*). Por ejemplo: Content Type = audio/MPEG.

Un flujo de medios puede identificarse por su localización y el protocolo usado para acceder a él. Se usa una URL para localizar un fichero remoto o local. Cuando no se puede usar la URL del fichero, se usa un localizador de medios que, de forma similar a la URL, proporciona el camino para identificar la localización del flujo de medios.

2.2.2 Principales clases e interfaces de la librería JMF

El API de la librería JMF consiste principalmente en interfaces y clases abstractas que definen el comportamiento de objetos usados para capturar, procesar, presentar y almacenar medios basados en tiempo.

A continuación se presentan algunas de las más relevantes para la comprensión del funcionamiento básico del *Framework* [JMFGUIA]:

DataSource

Un objeto de este tipo se encarga de encapsular la localización, el protocolo de transferencia y el software necesario para la adquisición del medio, es decir, del contenido multimedia.

Se construye a partir de una URL o un localizador de medios definido por la clase `MediaLocator` del API.

Los protocolos que usa esta librería para localizar el contenido multimedia son FILE para el contenido almacenado en local, o HTTP para el contenido que se encuentra en remoto.

Player y *Processor*

La función principal de un objeto de cualquiera de estos tipos es la de manejar los datos multimedia obtenidos del `DataSource`, para su procesamiento y presentación. En el siguiente apartado, “Fases del tratamiento de medios”, se hablará más acerca de ellos.

DataSink

El último paso en el procesado de los datos multimedia puede ser la renderización, el almacenamiento en algún fichero o la transmisión. Para estos dos últimos JMF proporciona la clase `DataSink`. Con un objeto de esta clase se puede obtener el flujo de medios y almacenarlo en un fichero local, o un fichero remoto o transmitirlo mediante RTP.

Para la creación de objetos de cualquiera de estas clases se usa un manejador implementado por la clase `Manager`.

2.2.3 Player

Por su importancia en el contexto de este proyecto, a continuación se describen algunas de las características más importantes de un objeto `Player`.

Un `Player` maneja el contenido multimedia de forma integral. Primeramente obtiene los datos a partir de un `DataSource` y posteriormente permite su Demultiplexión, Decodificación, y Renderización. En la siguiente figura se muestra un esquema de este proceso.

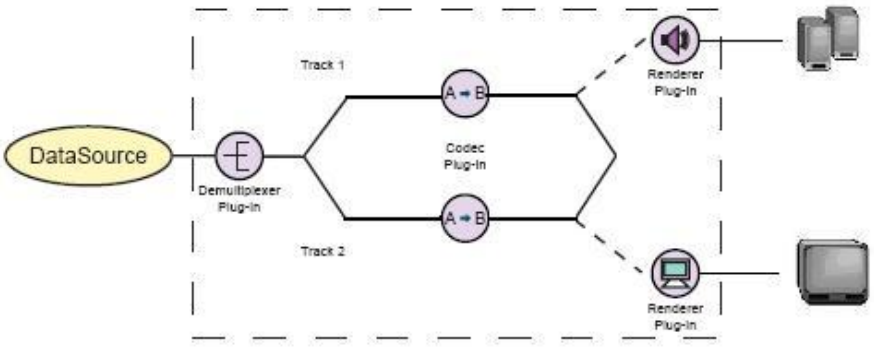


Figura 2.14. Etapas del procesamiento de un *Player* [JMFGUIA]

Es conveniente recordar, que si se tienen diferentes flujos de medios se ha de usar un `Player` para la reproducción de cada uno.

En el proceso que realiza un `Player` el usuario no puede intervenir. Como consecuencia, no soporta ningún control sobre el procesamiento que realiza, ni sobre cómo realiza la renderización de los medios.

Provee interfaces de control para el proceso de descarga en caché (*`Caching`*`Control`) y el control de la ganancia del volumen (*`GainControl`*). Por otra parte, dispone de métodos para establecer y obtener la tasa de reproducción y un punto temporal en la reproducción, y un método para la obtención de la duración del medio.

Proporciona un componente visual que representa la ventana de visualización del vídeo, en caso de que el flujo de medios lo contenga, y que podrá ser añadido como componente de la interfaz gráfica del reproductor.

Estados de un Player

Durante el proceso de reproducción de un flujo de medios, un `Player` se puede encontrar en cualquiera de los siguientes estados mostrados en la siguiente figura:

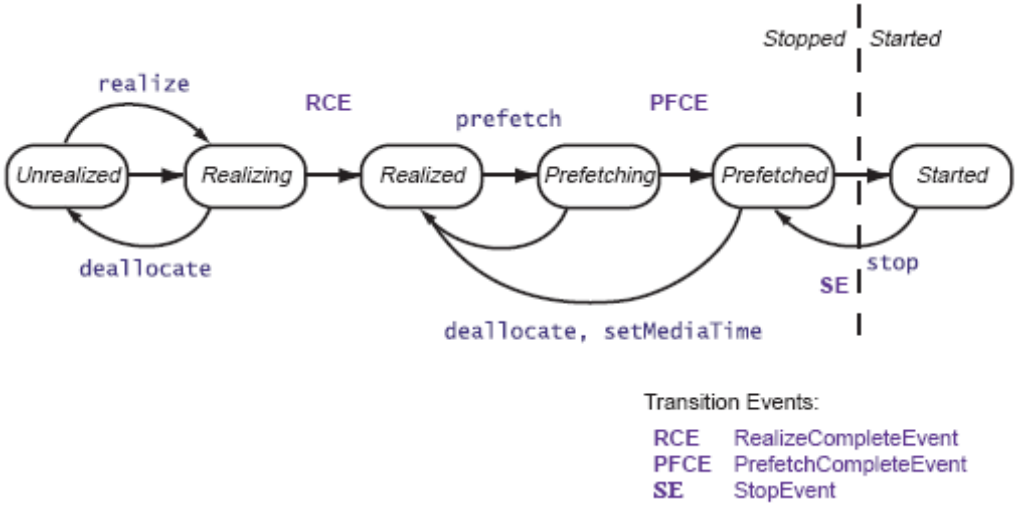


Figura 2.15. Estados del *Player* [JMFGUIA]

- *Unrealized*: el `Player` ha sido instanciado pero aún no conoce nada sobre su medio.
- *Realizing*: el `Player` está procediendo a averiguar los recursos que necesita para la reproducción del medio. Adquiere los recursos que necesita al principio, con excepción de aquellos que son de uso exclusivo por un `Player` al mismo tiempo. También es en este proceso donde se realiza la descarga en local, *caching*, del medio, en el caso de que el medio se encuentre en remoto y de estar disponible esta opción.
- *Realized*: el `Player` ha acabado el proceso de *realizing* y ya dispone del conocimiento acerca de los requerimientos de recursos y la información sobre el tipo de medios que va a presentar. En este momento, ya puede proporcionar los componentes visuales y de control. Las conexiones a otros objetos en el sistema están hechas, pero no posee ningún mecanismo que evitase que otro `Player` arrancara.
- *Prefetching*: el `Player` está preparado para presentar su medio. Durante esta fase se precargan los datos, obteniendo recursos de uso exclusivo, y se prepara para arrancar. El *Prefetching* debe repetirse si la presentación del medio se reposiciona, o se produce un cambio en la tasa de reproducción que requiera añadir buffers adicionales o que se den otros procesos alternativos en el `Player`.
- *Prefetched*: el `Player` ya está preparado para ser arrancado.
- *Started*: el `Player` ya ha arrancado. La referencia temporal de un `Player` (*Base Time*) y el tiempo de reproducción del medio (*Media Time*) son mapeados, y su reloj esta ya en ejecución.

Cuando el `Player` alcanza un determinado estado lanza un evento, `Tansition Event`. Implementando la interfaz `ControllerListener` se puede escuchar estos y otros eventos lanzados por el `Player`, y así permitir conocer en qué estado se encuentra y responder consecuentemente.

El conocimiento de los distintos estados por los que pasa el `Player`, así como el mecanismo de escucha de eventos de transición entre estados, se hace necesario a la hora de desarrollar código para la reproducción de medios con JMF.

2.2.4 Soporte a RTP en la librería JMF

Streaming: Protocolo RTP

Antes de proceder con el API de JMF para RTP, se presentará brevemente el protocolo RTP.

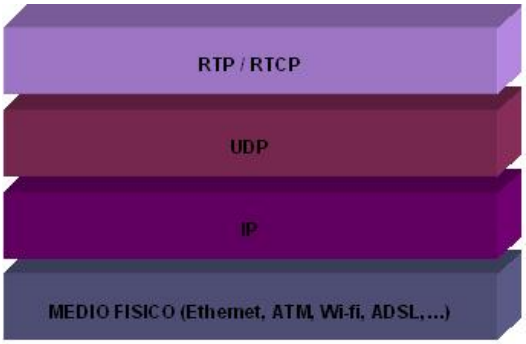


Figura 2.16. Torre de protocolos para RTP/RTCP

Como se muestra en la figura anterior, RTP (*Real Time Protocol*) es un protocolo de nivel de sesión definido sobre el protocolo de transporte UDP para la transmisión de flujos de datos multimedia en tiempo real sobre la red.

Surgió para dar solución a las limitaciones del protocolo HTTP para la realización de *streaming* de audio y video. HTTP no fue diseñado para este tipo de comunicaciones, requiriendo una alta disponibilidad de la red para que la reproducción resulte aceptable. Por ello, la mayoría de las aplicaciones consumidoras de contenidos alcanzables mediante HTTP realizan un paso previo de almacenamiento temporal en local, como el anteriormente mencionado *caching*.

RTCP es un protocolo asociado a RTP para el envío de datos de control. Los paquetes RTCP pueden contener información sobre calidad de servicio, sobre la fuente de los medios a ser transmitidos, y sobre estadísticas de la sesión RTP. Se envían paquetes de control periódicamente a todos los participantes de la sesión.

Una sesión RTP se define como una asociación entre un conjunto de aplicaciones distribuidas en una red y que se comunican con RTP. Una sesión se identifica por una dirección IP y un par de puertos, uno para el flujo de datos y el otro para el de control. El de control suele ser el inmediatamente posterior al de datos.

Especificación de JMF para RTP

JMF permite la trasmisión y recepción de flujos RTP a través de APIs definidos para ello. Esta especificación del API para RTP está diseñada para poder desarrollar fácilmente soluciones para el *streaming* RTP. [JMFGUIA]

Recepción:

Se pude recibir flujos RTP para su reproducción local, para guardarlo en un archivo, o procesarlo.

Los objetos `Player` y `Processor` son utilizados para manipular y presentar flujos de medios RTP como cualquier otro medio almacenado en fichero.

A continuación se muestra un esquema del procesado para la recepción de un flujo RTP:

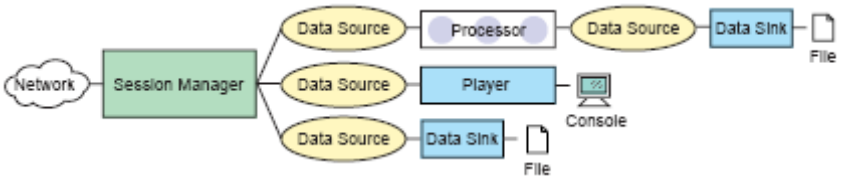


Figura 2.17. Esquema del proceso de recepción RTP [JMFGUIA]

Para la recepción y posterior presentación de un único flujo se puede crear un `Player` a partir de un localizador de medios del tipo:

rtp://address:port[:ssrc]/content-type/[ttl]

Donde,

address: es la dirección IP utilizada en la sesión
port: es el puerto donde se recibe el flujo de datos
ssrc: es un identificador de la fuente de datos
ttl: *time-to-life*, es el tiempo de vida por defecto de los paquetes de datos
(Los campos "ssrc" y "ttl" son opcionales, y en el caso de "ttl" se utiliza para *multicast*).

Para la recepción de uno o más flujos se utiliza un manejador de sesión por cada uno de ellos. Cuando se éstos se reciban, se crearán tantos objetos `Player` como flujos de medios se quieran presentar.

Transmisión:

Se pueden transmitir flujos de medios RTP almacenados en un archivo, o que hayan sido capturados por un dispositivo de captura local. Un `Processor` es el encargado de procesar los datos para adaptarlos a un formato RTP compatible. Posteriormente, un manejador de sesión se encargará de crear el flujo a enviar. La siguiente figura muestra un esquema de este proceso.

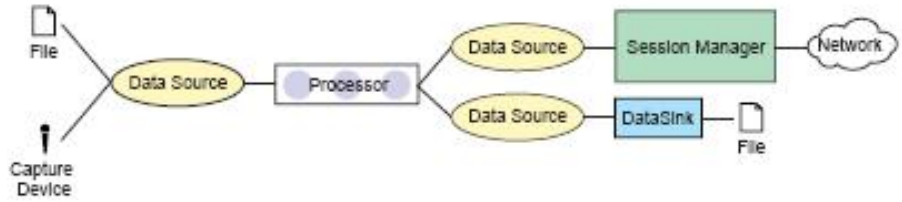


Figura 2.18. Esquema del proceso de transmisión RTP [JMFGUIA]

Manejador de la sesión RTP

Tanto en la transmisión como en la recepción, se ha de tener en cuenta que un manejador de sesión sólo puede manejar una única sesión. Si una aplicación maneja varias sesiones RTP, deberá crear un manejador de sesión por cada una de ellas.

Para la creación de manejadores de sesiones el API de JMF define una clase abstracta, `RTPManager`. Esta clase proporciona métodos para la creación, mantenimiento y cierre de una sesión RTP. Habilita a una aplicación a realizar funciones tales como iniciar la participación en una sesión o eliminar flujos individuales.

Por lo tanto, un manejador de sesión ha de extender esta clase sobrescribiendo sus métodos y añadiendo funcionalidad para controlar el estado de la sesión, los participantes y flujos activos, manejar los canales de control RTCP, y otras capacidades.

La implementación de referencia de la librería JMF, realizada por Sun Microsystems, proporciona una clase, que extiende de `RTPManager`, y que asume toda esta funcionalidad. Se trata de la clase `com.sun.media.rtp.RTPSessionMgr`. Para crear un manejador de sesión RTP basta con construir una instancia de ésta.

Creación de la conexión

El API define una interfaz, *`RTPConnector`*, que permite al programador abstraer al `RTPManager` del protocolo de transporte del flujo de datos y control de sobre el que se sitúe RTP. Implementando esta interfaz se puede enviar flujos RTP desde cualquier subcapa de red a través de *sockets* (puntos terminales de una conexión).

Durante la inicialización del `RTPManager` se debe crear y añadir al mismo una implementación de la interfaz *`RTPConnector`*.

En el caso del emisor, el `RTPManager` crea el stream RTP para que posteriormente sea transmitido a la red através del *socket* de salida creado para ello en la implementación del *`RTPConnector`*.

En el receptor, se espera a recibir un stream en el *socket* de recepción creado para ello en la implementación del *`RTPConnector`*, para posteriormente crear un `Player` a partir de él.

Eventos RTP

Si se desea averiguar el estado de la sesión, es decir, cuándo se incorpora o abandona un participante, cuando llega un nuevo stream, o cuando se ha producido una colisión, entre otros casos, se puede hacer uso de un mecanismo de notificación de eventos RTP que proporciona el API de JMF para RTP, al igual que ocurre con lo que se comentó en apartados anteriores sobre los eventos de los objetos `Controller`. Para poder recibirlos, se ha de implementar al apropiado escuchador, `Listener`, y registrarlo en el `RTPManager`.

2.2.5 Extender la librería JMF

La funcionalidad de esta librería puede ser extendida por medio de:

- La implementación de nuevos componentes de procesamiento (*plug-ins*), tales como *codecs*, *parsers*, *effects*,… que se puedan intercambiar con los ya existentes.
- La implementación directa de algunas de las interfaces que define el API, tales como *`Controller`*, `Player`, `Processor`, `DataSource` o `DataSink`.

2.2.6 Justificación del uso de JMF

Al tratarse de un *MediaRenderer*, se hace necesaria una tecnología que permita desarrollar fácilmente código para el procesado del contenido multimedia y su posterior renderización. El API de JMF abarca todo el proceso, desde la obtención del medio hasta su renderización, pasando por una fase de procesamiento que incluye la decodificación de los datos.

La aplicación se ha desarrollado íntegramente en lenguaje Java, por lo que para mantener una cierta coherencia, se debía usar una librería multimedia que estuviese implementada en este lenguaje. La librería multimedia y multiplataforma de referencia en Java es, a día de hoy, JMF.

Su sencillez y extensibilidad permite crear código para el tratamiento de medios multimedia con cierta facilidad y poder extender su funcionalidad de la misma forma.

2.3 OSGi

En las próximas líneas de este apartado se estudiará la tecnología OSGi. En primer lugar, se hará una breve introducción para presentar las bases conceptuales de la misma. A continuación se enumerarán las características de esta especificación y se describirá la arquitectura software que define. Al igual que en los anteriores apartados del capítulo, en último lugar se justificará el uso de la tecnología en este proyecto.

2.3.1 Introducción

La tecnología OSGi, de las siglas en inglés *Open Service Gateway Initiative*, especifica una plataforma basada en Java para el desarrollo de aplicaciones y servicios. Su objetivo es el de proporcionar una tecnología que asegure la interoperabilidad entre estas aplicaciones y servicios y su gestión remota a través de la red.

Está promovida por la *OSGi Alliance*. Se trata de una organización creada en el año 1999 para estandarizar los esfuerzos de conexión de un amplio rango de aplicaciones y dispositivos a un entorno residencial. Está compuesta por compañías expertas en el sector tecnológico entre las que se encuentran conocidas compañías de tecnología móvil, informática, y desarrollo software. [OSGi]

Define un *Framework* Java, a través de un conjunto de APIs, cuya principal aportación es que dota de modularidad dinámica a Java. De esta forma, una aplicación Java puede ser construida a partir de pequeños módulos pre-compilados, reutilizables y compartidos por otras aplicaciones, y a su vez se le proporciona la capacidad de poder ser gestionada dinámica y remotamente.

Está orientada a la capa de aplicación, lo que permite al desarrollador de un servicio o aplicación abstraerse de aspectos subyacentes como el sistema operativo y el hardware de la plataforma, las tecnologías de red o los protocolos de comunicación.

En una plataforma de servicios OSGi, todos los servicios disponibles pueden ser utilizados por las aplicaciones y/o otros servicios. A si mismo, una aplicación puede exportar a la plataforma servicios propios para poder también ser compartidos.

Los tipos de dispositivos donde se puede desplegar esta plataforma son todos aquellos que representan los diversos mercados en los que los que los miembros de la alianza desarrollan su actividad, tales como pasarelas de servicios, operadores de infraestructuras de red, desarrollo software, automóviles, móviles y un largo etcétera.

Entre estos dispositivos cabe destacar, por su interés en el contexto de este proyecto, la Pasarela de Servicios. También llamada Pasarela Residencial o Pasarela Doméstica cuando se refiere al ámbito de la vivienda. Este dispositivo conecta toda la infraestructura de telecomunicaciones (datos, control, automatización, entretenimiento…) del hogar digital a una red pública de datos, como Internet. Entre sus funciones están las de *router*, *hub*, módem, cortafuegos, y plataforma de ejecución de aplicaciones típicas para el hogar, ya sean de entretenimiento, de vigilancia, de teleasistencia, e-comercio o control domótico. En este sentido, la especificación OSGi proporciona a la pasarela un entorno software para el desarrollo de este tipo de servicios y aplicaciones que pueden ser desplegadas por la red y gestionadas remotamente.

La siguiente figura muestra el escenario típico en el que una pasarela de servicios es utilizada para interconectar las redes de una vivienda, y conectada con Internet permitiendo el manejo remoto de la misma.

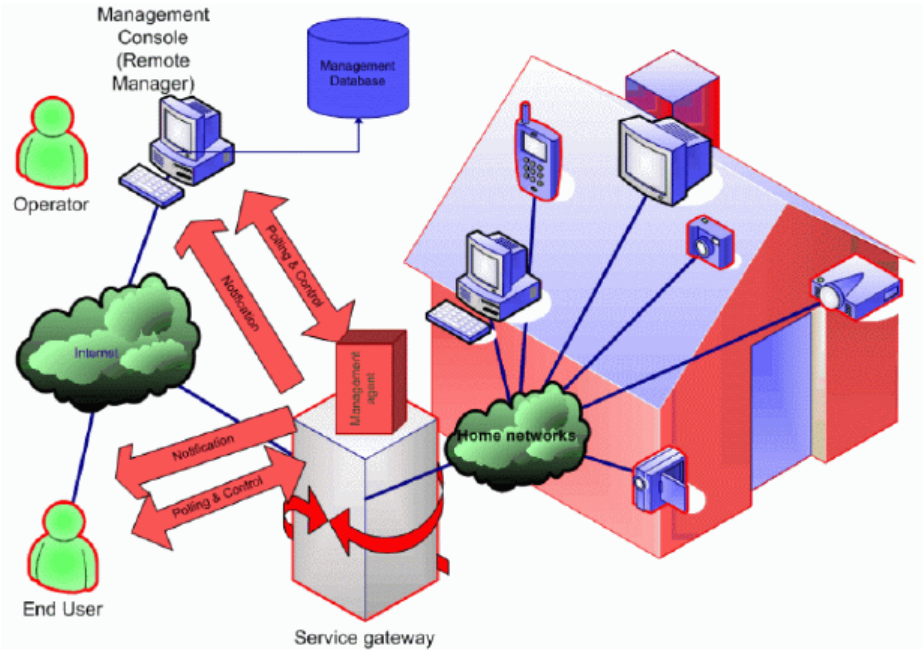


Figura 2.19. Pasarela de Servicios OSGi (Pasarela Doméstica)

2.3.2 Características de la especificación

Las características generales de la especificación OSGi son:

- **Portable:** las APIs de OSGi pueden ser implementados en un gran rango de plataformas hardware y sistemas operativos, pudiéndose adaptar a múltiples soluciones en todos los ámbitos.
- **Segura:** incorpora un modelo de seguridad que cubre varios niveles, permitiendo proporcionar un entorno de ejecución seguro.
- **Estándar:** ofrece una plataforma común para todos los fabricantes de dispositivos, desarrolladores y proveedores de servicios para facilitar el avance de la tecnología y a su vez evitar los monopolios.
- **Abierta:** no obliga al uso de ninguna tecnología determinada. Únicamente se deben desarrollar aplicaciones y servicios compatibles con las APIs definidas.
- **Escalable:** permite a los proveedores de sistemas personalizar y administrar fácilmente la plataforma.
- **Dinámica:** permite la actualización de los componentes software, pudiendo adaptarse a las necesidades específicas del usuario, y sin tener que reiniciar el sistema.
- **Fiables:** la plataforma debe estar operativa las 24 horas del día.

2.3.3 Arquitectura software

El *Framework* OSGi proporciona un entorno de ejecución para aplicaciones y servicios, programados en Java.

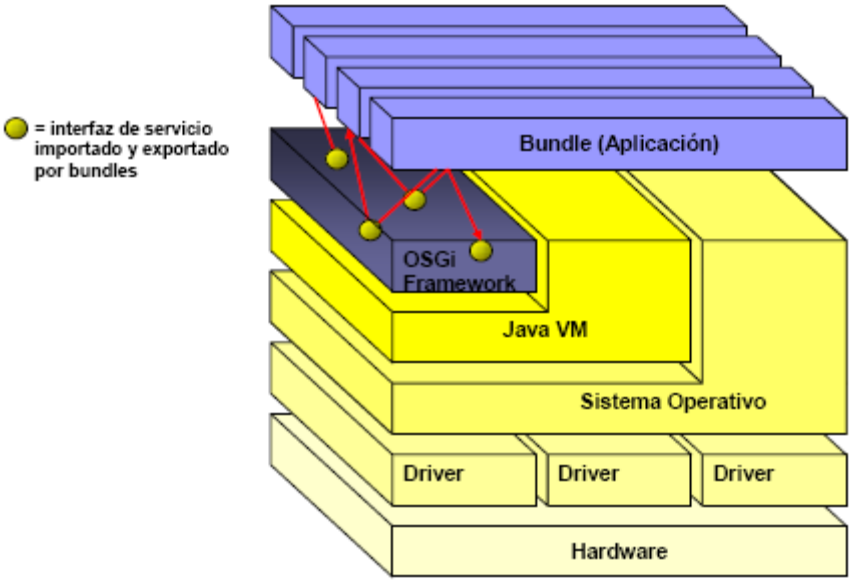


Figura 2.20. Capas de la Arquitectura Software de OSGi [OSGiAPUNTES]

Como se puede ver en la Figura 2.20, la base de su arquitectura software de OSGi es una Máquina Virtual Java. Características del entorno de desarrollo Java, tales como, la seguridad, la fiabilidad, la apertura, y sobre todo, la independencia de la plataforma, justifican este uso por guardar perfecta coherencia con los objetivos de esta tecnología.

Dentro de esta arquitectura software se pueden distinguir dos importantes entidades, los *bundle*s y los servicios.

Bundle

Los *bundle*s son archivos empaquetadores Java (jar), compuestos por un conjunto de clases (Java) y otros recursos necesarios para proporcionar funcionalidad a los usuarios finales y exportar capacidades, denominadas Servicios, para que otros *bundle*s puedan utilizarlos [OSGiSPECIFIC].

Son las entidades OSGi para el despliegue de aplicaciones Java. En este sentido, se pueden construir una aplicación en base a los servicios contenidos en el *bundle*, o bien, se puede integrar una aplicación completa dentro de un único *bundle*.

Entre el conjunto de clases privadas contenidas en el jar, debe haber una que implemente la interfaz `BundleActivator`, definida por el API de OSGi, donde necesariamente se deben implementar dos métodos `start` y `stop`, llamados al arrancar y terminar el *bundle*. El método `start` es utilizado para registrar servicios y localizar cualquier recurso que use el *bundle*. El método *stop* es utilizado para parar cualquier actividad, hilo de proceso que haya creado el método `start`.

Dentro del mismo archivo debe incluirse un fichero de texto, denominado MANIFEST, en el que figuran un conjunto de cabeceras que contienen la información que el *Framework* necesita para que el *bundle* se instale y ejecute correctamente. Se describen a continuación:

Bundle-Activator: Nombre de la clase que implementa la interfaz `BundleActivator` y que sirve para iniciar y detener el *bundle*.

Bundle-Category: Lista de categorías de *bundle*s a las que pertenece este mismo *bundle*. Esto permite que los *bundle*s sean listados por su categoría.

Bundle-ClassPath: Nombre y localización de los ficheros JAR y otros recursos necesarios para la ejecución del *bundle*.

Bundle-ContactAddress: Dirección de contacto del fabricante del *bundle*.

Bundle-Copyright: Copyright del *bundle*

Bundle-Description: Información adicional sobre la funcionalidad y servicios del *bundle*.

Bundle-DocURL: Dirección URL donde se encuentra más información acerca del *bundle*.

Bundle-Name: Nombre del *bundle*. Este debe ser corto y sin espacios. Es obligatoria.

Bundle-NativeCode: Especificación de librerías de código nativo contenidas en el *bundle* y usadas por este.

Bundle-BundleRequiredExecutionEnvironment: Entornos de ejecución requeridos que deben estar presentes en la plataforma para la ejecución del *bundle*.

Bundle-UpdateLocation: Ubicación que se debe tener en cuenta para la actualización del *bundle*.

Bundle-Vendor: Nombre del fabricante del *bundle*.

Bundle-Version: Versión del *bundle*.

Export-Package: Lista de paquetes que deben ser exportados para ser compartidos con otros *bundle*s en el *Framework*. Opcionalmente se puede indicar la versión del paquete.

Import-Package: Lista de paquetes que deben ser importados para la correcta ejecución del *bundle*. Los paquetes importados haber sido exportados previamente de otros *bundle*s. Opcionalmente se puede poner la versión del paquete que debe ser importado.

DynamicImport-Package: Lista de paquetes que deben ser importados dinámicamente, es decir, cuando el código en ejecución lo precise.

Estas cabeceras están definidas por OSGi y son todas opcionales, excepto *Bundle-Name*.

Es importante resaltar que el *Framework* OSGi define un contexto de ejecución propio para el *bundle* y habilita un área de almacenamiento persistente para los recursos que éste necesita. Es decir, se ha de utilizar este contexto de ejecución para poder acceder a recursos, como puede ser un fichero, que el *bundle* necesita. Este contexto viene definido por la interfaz `BundleContext` que provee el API de OSGi.

Ciclo de vida de un bundle

El sistema de gestión de la plataforma permite que un bundle se descargue, se instale, se ejecute, se detenga, se desinstale y se actualice, y todo sin tener que reiniciar el sistema.

Por tanto, un *bundle* puede encontrarse, en alguno de los siguientes estados:

- **Instalado (*INSTALLED*):** cuando se instala exitosamente en la plataforma y conoce los recursos que necesita, definidos en el manifiesto.
- **Inactivo (*RESOLVED*):** todas las clases del *bundle*, incluidas las importadas, están disponibles y, por consiguiente, está preparado para ejecutarse.
- **Funcionando (*STARTING*):** el *bundle* ha sido iniciado. Se ha llamado al método *start* de la clase que implementa la interfaz `BundleActivator` del mismo.
- **Activo (*ACTIVE*):** el *bundle* se ejecuta exitosamente.
- **Parándose (*STOPPING*):** el *bundle* deja de ejecutarse. Se ha llamado al método *stop* de la clase que implementa la interfaz `BundleActivator` del mismo.

- **Desinstalado (*UNINSTALLED*):** el *bundle* ha sido desinstalado

La figura que se muestra a continuación representa este ciclo de vida.

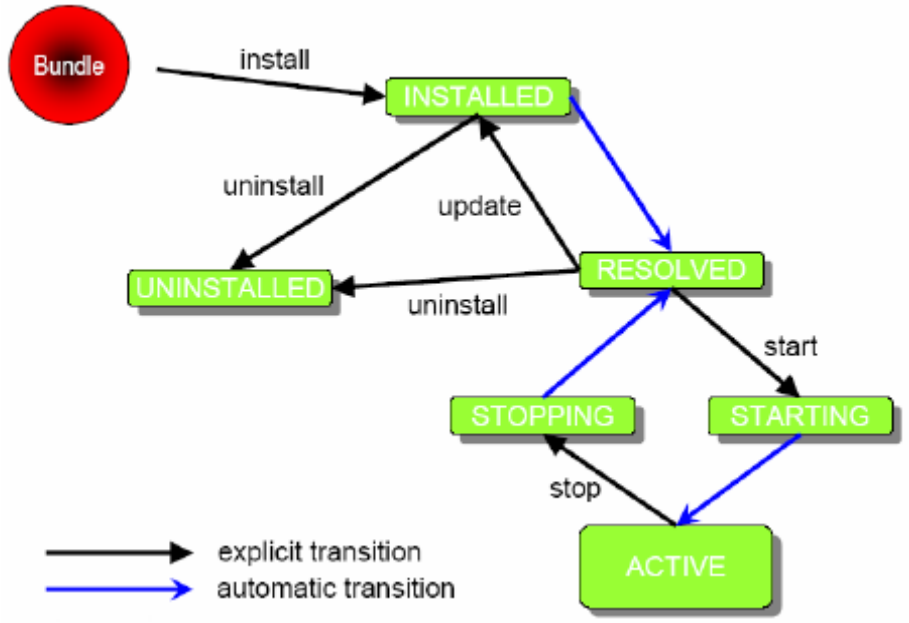


Figura 2.21. Ciclo de vida de un *bundle* [OSGiAPUNTES]

Servicio

Un Servicio es un objeto que proporciona funcionalidad. Es registrado por un bundle en el *Framework* OSGi pare ser usados por otros *bundles*.

Como se dijo anteriormente, un *bundle* puede extender sus capacidades implementando servicios disponibles en el *Framework*, y también, puede exportarlas registrando servicios para que otros *bundle*s puedan usarlos.

Para ello, el *Framework* de OSGi proporciona un Servicio de Registro, *OSGi Service Registry*, que habilita a los *bundle*s a:

- Registrar servicios.
- Buscar servicios en el registro.
- Recibir notificaciones de cuando un servicio esta registrado o deja de estarlo.

La semántica y sintaxis de un servicio se especifica en un interfaz Java. En él se declaran los métodos que ofrece el objeto que representa el servicio. Para que un *bundle* asuma las capacidades de este debe implementar ésta interfaz. De esta forma, se hace una definición abstracta de los servicios, y son los fabricantes y desarrolladores los que los implementan de forma independiente.

Los *bundle*s pueden acceder a los servicios disponibles en el *Framework* por medio del `BundleContext`.

El `BundleContext` representa el entorno de ejecución, específico del *bundle*, que se crea cuando éste se instala en el *Framework*. A través de este se pueden realizar las mencionadas operaciones de registro de servicios, obtención de servicios registrados, así como agregar o eliminar las suscripciones para la recepción de eventos asociados a estos servicios.

La especificación OSGi ha implementado varios Servicios Estándar que pueden ser implementados por cualquier *bundle* y registrarlos por estos en el Registro de Servicios. [OSGiSERVICES]

- **Servicios del *Framework*:** para dirigir las operaciones en el *Framework* OSGi.
- **Servicios del Sistema:** para proveer de funcionalidad común a prácticamente todos los sistemas donde se implemente OSGi.
- **Servicios de Protocolo:** para mapear un protocolo externo en un servicio OSGi, integrándole en la plataforma.
- **Otros:** y otros servicios para funcionalidades específicas.

A continuación se muestra un esquema de los servicios estándar proporcionados por la versión 4 de la especificación:

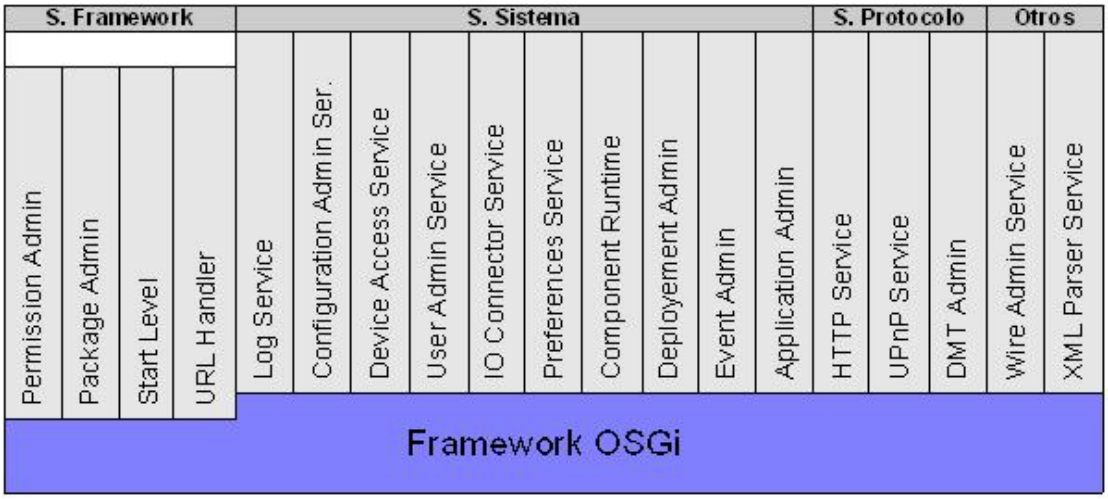


Figura 2.22. Esquema de los Servicios Estándar de OSGi

Se puede ampliar esta información sobre los Servicios Estándar de OSGi en la documentación de la especificación disponible en la web oficial [OSGiSPECIFIC].

2.3.5 Justificación del uso de OSGi

La especificación OSGi constituye la tecnología de un futuro, cada día más cercano, para integrar los múltiples servicios y aplicaciones ofrecidos a los consumidores a través de Internet o redes móviles. De este modo, servicios como el telecontrol, la video-vigilancia, la telemedicina, la domótica, o el entretenimiento digital, pueden se gestionados de forma segura, fácil y con independencia de las tecnologías de red, a través de una plataforma común.

La implementación de esta plataforma como pasarela de servicios en el ámbito de la vivienda, permite la interconexión de múltiples dispositivos hardware y software que hacen uso de estos servicios y aplicaciones. De esta forma se convierte en el núcleo del futuro Hogar Digital [OSGiCASADOMO].

Por ello, en el desarrollo de una aplicación software creada para el entretenimiento digital en el hogar, es oportuno considerar éstas tendencias y buscar la compatibilidad con esta tecnología. En consecuencia, la aplicación se integra en un *bundle* OSGi, para así poder ser instalada y ejecutada en pasarelas de servicios conformes con este estándar.

2.4 Resumen de la utilización de las tecnologías

OSGi, esencialmente, proporciona el control del ciclo de vida de la aplicación permitiendo su ejecución en dispositivos avanzados para la gestión del hogar digital como las Pasarelas de Servicios.

UPnP define cómo la aplicación puede agregarse a una red (UPnP) y cómo puede ser controlada remotamente usando mensajes XML enviados sobre HTTP. En base a este modelo de

comunicación entre dispositivos UPnP, la Arquitectura para Audio y Video de UPnP, proporciona la capacidad de compartir cualquier contenido multimedia localizado en la red doméstica y su reproducción por medio de dispositivos de renderización distribuidos por la misma red como el implementado en el presente proyecto. Todo esto controlado por el usuario a través de un único dispositivo, el *Control Point*.

JMF provee un catálogo de métodos, clases e interfaces Java con las que poder desarrollar el código necesario en la aplicación para manejar el contenido multimedia desde su obtención hasta su renderización.

2.5 Estado del arte

Tras describir las principales tecnologías sobre las que se fundamenta este proyecto, y profundizar en los conceptos teóricos de la especificación UPnP para el *Media Renderer*, se puede proceder a evaluar la situación actual de este tipo de dispositivos presentando las soluciones disponibles en el mercado. Tras esto, se analizará de forma breve el estado de la librería JMF y se enunciarán posibles alternativas a la misma que puedan salvar sus principales limitaciones. Por último, se enumerarán algunas de las soluciones software que implementan la plataforma OSGi en la actualidad.

2.5.1 Implementaciones UPnP MediaRenderer

En la actualidad existen múltiples dispositivos conformes con el estándar UPnP. Se pueden encontrar varias soluciones software y hardware de *AV Control Point* y *MediaServer*s.

En cuanto a dispositivos *UPnP MediaRenderer* existen también varias soluciones hardware pero, sin embargo, no es fácil encontrar una aplicación software que cumpla la especificación. Entre las soluciones obtenidas que la cumplen no se ha encontrado ninguna que reuniese capacidades tales como multiplataforma, integración en plataformas OSGi o soporte específico para IPv6. Capacidades que por otro lado son marcadas como características generales del prototipo a desarrollar, tal y como se mencionó en la definición de objetivos del presente proyecto.

A continuación se listan las soluciones, software y hardware, que se han encontrado:

Software:

- GMediaRender: se trata de una aplicación de código abierto para únicamente plataformas Linux. Está implementada en lenguaje C, y basada en la librería multimedia GStreamer y en la librería *Portable* SDK *for UPnP Devices* (libupnp) para el desarrollo de dispositivos compatibles con UPnP. En la actualidad se encuentra en su versión 0.0.6 de 2007. [GMRENDER]

- MPlayer: es otra aplicación de código abierto y programada en C. Para la compatibilidad con UPnP disponde de un componente, aún en el estado inicial de su desarrollo, denomindo UPnP-Control implementado también mediante el uso de la librería “libupnp”. Al igual que en el caso anterior sólo está disponible para las plataformas Linux. Aunque la versión actual de la aplicación, la 1.0, es del mismo año que el de la ejecución del presente proyecto, no ocurre lo mismo con el módulo para UPnP, cuya última actualización se remonta al año 2006. [MPLAYER]

- Compère de Coherence: es una aplicación, también de código abierto, que combina un *UPnP Control Point* y un *UPnP MediaRenderer*. Está programada en lenguaje Python y diseñada para ser integrada exclusivamente en el terminal móvil Nokia N800 *Internet Tablet*. [COMPERE]

- CyberLink PowerCinema: es una aplicación multimedia avanzada y de código cerrado para PCs con plataformas Windows. Puede actuar tanto de *UPnP MediaServer* como de *UPnP MediaRenderer*.[POWERCINEMA]

Hardware:

- DLink-DSM 320/520/750, entre otros: se trata de una serie comercial de dispositivos denominados *Media Players* inalámbricos, cada uno de los cuales conforman un *UPnP MediaRenderer*.

- Philips SL300/SL300i/SL400/SL400i, entre otros: se trata de otra serie comercial de dispositivos completamente análogos a la anteriormente presentada.

- Roku SoundBridge: es un dispositivo reproductor de audio conforme a la especificación UPnP para *Media Renderers*. Dispone de su propio software para el control de la reproducción desde un PC. Fuera de Estados unidos, se comercializa por medio de la marca Pinnacle.

También existe cierta variedad de soluciones que permiten listar el contenido multimedia compartido en una red UPnP y reproducirlos, pero no permiten ser controlados remotamente y en algunos casos usan sus propios protocolos de transferencia para obtener el contenido. Se podría decir que integran la capacidad de un *UPnP AV Control Point* de listar el contenido de un *UPnP MediaServer* en un reproductor multimedia para su reproducción. En algunos casos, a este tipo de dispositivo se les denomina *UPnP Player*. Pero lógicamente no cumplen la especificación de un *UPnP MediaRenderer*.

Ejemplos de este tipo de reproductores son:

Software:

- Nero 9.0: esta aplicación de código cerrado implementa esta capacidad por medio del módulo ShowTime. Sólo está disponible para plataformas Windows. [NERO]

- VLC 1.0: la última versión del reproductor de código abierto VLC, pretende incorporar este tipo de capacidad por medio del módulo UPnP Player. Está programado en lenguaje C y disponible para varias plataformas Windows, Linux, Mac, Solaris, etc. [VLC]

- XBMC *Media Center*: es un programa de código abierto y programado en C, que provee una solución para compartir el contenido multimedia distribuido en el hogar y su reproducción por medio del estándar UPnP. Está disponible para varias plataformas Windows, Linux y Mac. [XBMC]

- Pocket Player: se trata de un software de código cerrado creado para poder listar y reproducir el contenido multimedia disponible en una red UPnP. Está desarrollado únicamente para dispositivos limitados, PocketPCs y SmartPhones. [POCKETPLAYER]

Hardware:

- Videoconsola Xbox 360: permite la reproducción de contenido almacenado en *UPnP MediaServers* por medio del software anteriormente mencionado XBMC *Media Center* para PCs.

- Videoconsola PlayStation 3: este dispositivo permite listar el contenido localizable por *UPnP MediaServers* y reproducirlo.

- Terratec Noxon iRadio: se trata de un dispositivo con la misma capacidad que el caso anterior pero únicamente para el contenido de audio.

Por otra parte, existen kits de desarrollo para la creación de dispositivos UPnP. Se tratan de APIs de programación que facilitan el desarrollo de este tipo de aplicaciones.

Ejemplos de estos son:

- Platinum UPnP SDK [SDKPLATINUM]

- *Portable* SDK *for UPnP Devices* (libupnp 1.6.6) [SDKLIBUPNP]: las aplicaciones de software libre GMediaRender y MPlayer, utilizan esta librería para el desarrollo de sus capacidades UPnP.

2.5.2 JMF y alternativas

Como se dijo en su descripción, JMF ha venido siendo hasta estos días la referencia dentro de las librerías Java para el desarrollo de aplicaciones multimedia.

En el año 2003, Sun Microsystems, el precursor de esta tecnología, decidió dejar de dar soporte a esta librería [JMFOUT]. Sin embargo, se sigue pudiendo descargar la última versión del *Framework* desde su web oficial y se mantienen todos los recursos, desde documentación hasta su sección dentro del foro general de Sun [JMFFORO]. No en vano, a día de hoy, la participación en este foro indica que se sigue utilizando como librería de referencia.

El problema inmediato es el desfase tecnológico derivado de la continua aparición de nuevos formatos de audio y video, que obviamente JMF no soporta.

A continuación se muestran las tablas de los formatos comunes soportados por JMF [JMFGUIA].

Formato Soportado	**Tipo de *Codec***	**Formato Contenedor**	**Requerimiento de CPU**
Cinepack	Video	AVI QuickTime	Baja
MPEG-1	Video	MPEG	Alta
JPEG	Video	AVI QuickTime	Alta
H261	Video	AVI	Media
H263	Video	AVI QuickTime	Media
PCM	Audio	AVI QuickTime WAV	Baja
ADPCM (DVI)	Audio	AVI QuickTime WAV	Media
ADPCM (IMA4)	Audio	AVI QuickTime WAV	Media
GSM	Audio	WAV	Baja
G.711 (u-law)	Audio	AVI WAV QuickTime	Baja
Mu-law	Audio	AVI WAV QuickTime	Baja

G.723	Audio	WAV	Media
MPEG Layer 1,2	Audio	MPEG	Alta

Tabla 2.7. Formatos soportados por JMF

Tipo de Medio	RTP Payload
Audio: G.711 (U-law) 8 kHz	0
Audio: GSM mono	3
Audio: G.723 mono	4
Audio: 4-bit mono DVI 8 kHz	5
Audio: 4-bit mono DVI 11.025 kHz	16
Audio: 4-bit mono DVI 22.05 kHz	17
Audio: MPEG Layer I, II	14
Video: JPEG (4:2:0, 4:2:2, 4:4:4)	26
Video: H.261	31
Video: H.263	34
Video: MPEG-I	32

Tabla 2.8. Formatos RTP soportados por JMF

En vista de esta limitación en cuanto a formatos soportados, se pueden estudiar otras alternativas Java a JMF. Entre ellas se pueden encontrar las siguientes:

- *Freedom for Media in Java* (FMJ) [FMJ]: se trata de una librería multimedia Java basada en la de JMF. De hecho es compatible con su API. Soporta más formatos mediante *plug-ins*, al igual que JMF, y sobre todo por medio de la utilización de envoltorios (*wrappers*) para la utilización de librerías de *codecs* nativas como FFMPEG. El problema de esta librería es que no parece estar completa y su uso no está muy extendido.

- *Java Media Component* (JMC) [JMC]: es una nueva librería de Sun Microsystems para incorporar capacidades de procesado simple multimedia a las aplicaciones Java. Podría convertirse en la alternativa, quizás algo más limitada, de la propia librería JMF. Sin embargo, aún no hay suficiente información, y sólo se pude conseguir a través de la descarga del SDK (*Software Development Kit*) de JavaFx.

- JavaFX: es una tecnología lanzada recientemente por Sun Microsystems cuya finalidad es proveer de un conjunto de herramientas para facilitar el desarrollo de aplicaciones web con capacidades de aplicaciones de escritorio, como las aplicaciones multimedia. Las clases de JavaFx para el desarrollo de aplicaciones multimedia están basadas en JMC. El uso de ésta tecnología podrá ser interesante en futuros proyectos. A día de hoy hay cierta confusión sobre cómo se relacionan las clases Java comunes con la clases JavaFx, además su reciente lanzamiento provoca que surjan continuos bugs que deberán ser tratados.

- JVLC: permite la integración del reproductor nativo VLC en aplicaciones Java a través de la librería Java JNA (*Java Native Access*) para el acceso a código nativo. JVLC provee un conjunto de métodos y clases Java que están vinculados con el código nativo de VLC. Por tanto es dependiente de la versión de VLC instalada. Desde hace varios meses se retiraron los sitios web oficiales de esta tecnología, lo que complica su utilización. Además, el conjunto de métodos disponibles para el procesamiento multimedia podrían resultar limitados.

- JMF + *plug-ins*: a pesar de no poder utilizarse el código fuente de la librería JMF por estar sujeto a una licencia SCSL de Sun [SUNSCSL] que restringe su uso, gracias a la extensibilidad y modularidad proporcionada por los *plug-ins* JMF que se pueden integrar en el *Framework*, resulta relativamente sencillo ampliar el número de *codecs* soportados. Esto es lo que se consigue con soluciones como:

    - Fobs4JMF [FOBS]: *plug-in* de Fobs que actúa como envoltorio de la librería FFMPEG de *codecs*, permitiendo dar soporte a formatos como ogg, theora, divx, xvid, h264, etc.
    - Jffmpeg [JFFMPEG]: se trata de otro *plug-in* que envuelve la librería FFMPEG. En este caso su comportamiento con los ficheros de video no resulta aceptable.
    - mp3plug-in [MP3PLUG-IN]: *plug-in* creado por Sun Microsystems para el soporte del conocido formato de audio MP3 (MPEG-1 *Layer* 3).

A pesar de las limitaciones mencionadas de JMF, se optó por utilizar esta librería conjuntamente con *plug-ins* de *codecs*, por continuar siendo a día de hoy la librería multimedia Java de uso más extendido. También por su extensibilidad a través de *plug-ins*, y por toda la documentación y soporte a través de los foros de Sun que aún está disponible.

2.5.3 Implementaciones software de OSGi

Existen múltiples implementaciones de la plataforma OSGi en la actualidad. En la lista que se presenta a continuación se muestran las más conocidas de entre las disponibles:

Propietarias

- *mBedded Server de Prosyst* [PROSYST]: en su versión más actualizada implementa la especificación OSGi *Release* 4. Existen varias ediciones del producto en función de los distintos entornos de desarrollo:

    - *mBedded Server Equinox Edition*: de código abierto. Basado en el *Framework* de OSGi que proporciona Equinox (Eclipse) y al que añade nuevas características. Destinado al desarrollo de proyectos de código abierto.

    - *mBedded Server Professional Edition*: apropiado para el desarrollo de pequeñas plataformas en dispositivos con recursos limitados.

    - *mBedded Server CLDC Edition*: apropiado para la implementación de la plataforma en dispositivos, con recursos muy limitados, con Máquinas Virtuales Java compatibles con la configuración Connected Limited Device Configuration (CLDC) (implementa OSGi R3)

    Sobre las dos primeras ediciones se especifican tres distintas extensiones:

    - *mBS Smart Home Extension*: extensión orientada al desarrollo de la plataforma en pasarelas domésticas.

    - *mBS Telematics Extension*: extensión orientada principalmente al desarrollo de plataformas para el sector automovilístico.

    - *mBS Mobile Extension*: extensión orientada al desarrollo de la plataforma en dispositivos móviles.

- *Lotus Expeditor Client Platform* de IBM [EXPEDITOR]: esta plataforma implementa la especificación OSGi R4. Proporciona una solución completa y escalable para el despliegue, mantenimiento y eliminación de aplicaciones y componentes de aplicaciones sobre multitud de dispositivos.

De código abierto

- *mBedded Server Equinox Edition*: visto anteriormente.

- *Knopflerfish* de Makewave (anterior *Gatespace Telematics AB*) [KNOPFLERFISH]: es una implementación abierta, en la actualidad del *Realease* 4 de OSGi, que permite la gestión de las aplicaciones de una forma más visual e intuitiva que las otras implementaciones. En estos momentos se encuentra en su versión 2.3.

- *Felix* de la fundación Apache [FELIX]: implementa la especificación OSGi R4. Suministra un *Framework* OSGi completo. Actualmente se encuentra en su versión 1.8.

- *Equinox* desarrollado por la comunidad Eclipse [EQUINOX]: implementa la especificación OSGi R4. Es el responsable de los desarrollos OSGi de Eclipse. Abarca todos los sectores donde implementar OSGi (Telefonía Móvil, Hogar y Automoción)

- *Newton Project* de Paremus [NEWTON]: es un *Framework* OSGi distribuido basado en Service Component Architecture (SCA). Proporciona la capacidad de desplegar y gestionar dinámicamente servicios y *bundle*s de OSGi y los distribuye entre diferentes recursos. Actualmente se encuentra en su versión 1.3.2.

3. Implementación

En este capítulo se aborda el diseño de un *UPnP MediaRenderer* en base a los criterios establecidos por las distintas tecnologías que se han ido definiendo en el capítulo anterior.

Para facilitar la compresión de este capítulo se han utilizado varios diagramas UML que sintetizan de forma gráfica las características y arquitectura del sistema diseñado. En el “ANEXO III” se explica la simbología de estos diagramas.

Como paso previo se analizarán los requisitos del sistema y se expondrán los Casos de Uso del mismo. Tras esto, se enumerarán las herramientas utilizadas para el proceso y finalmente se describirá la implementación obtenida.

3.1 Análisis de requisitos

En primer lugar se ha de analizar los requisitos previos que se cumplirán. En este sentido, se definieron un conjunto de requisitos, tanto funcionales como no funcionales, que la implementación debía cumplir en base a los objetivos marcados al inicio del proyecto. Estos son:

Requisitos funcionales:

1. *Compatibilidad con UPnP*: la aplicación ha de cumplir con los protocolos que define el estándar UPnP. Permitiendo el control de la misma por el Punto de Control o *Control Point* e implementando los tres servicios definidos para un *UPnP MediaRenderer*. [Obligatorio]

2. *Control de la reproducción*: se debe proporcionar soporte para las acciones comunes de un reproductor multimedia, tales como, Arrancar, Pausar, Ir al siguiente ítem, Ir al anterior ítem, Parar, Subir y Bajar el volumen, o Silenciar el audio. [Obligatorio]

3. *Visualización de imágenes*: se debe proporcionar soporte para la visualización de imágenes. [Obligatorio]

4. *Reproducción de flujos RTP de audio y video*: se debe proporcionar soporte para el manejo y reproducción de flujos RTP de audio y video (*streaming* RTP). [Obligatorio]

5. *Reproducción de flujos HTTP de audio y video*: se debe proporcionar soporte para la reproducción de contenido de audio y video obtenido a través del protocolo HTTP. [Obligatorio]

Requisitos no funcionales:

1. *Compatibilidad con IPv6*: la aplicación debe ser compatible con el protocolo de red IPv6, para poder ser utilizada en escenarios con redes IPv6. [Obligatorio]

2. *Multiplataforma*: se ha de buscar una solución multiplataforma. Principalmente, la aplicación ha de poder ser ejecutada en las dos plataformas más comerciales, Linux y Windows. [Obligatorio]

3. *Integración en OSGi*: la aplicación debe integrarse en una plataforma OSGi, por lo que ha de ser conforme con la última versión del estándar, OSGi *Release* 4. [Obligatorio]

4. *Reproducción de los formatos más comunes*: se ha de buscar una solución que pueda soportar los formatos de codificación más comunes tanto de audio y video como de imágenes. [Obligatorio]

5. *Extensibilidad*: se pretende realizar un diseño que permita extender la funcionalidad de la aplicación, ya sea añadiendo *plug-ins* de codificación o añadir nuevos servicios UPnP. [Opcional]

3.1.1 Casos de uso

Tras haber definido los requisitos del sistema se pude generar el Diagrama de Uso del mismo. Este diagrama pretende representar de forma gráfica y sencilla la funcionalidad de la aplicación y su interacción con los actores que intervienen en el proceso.

Se han definido tres actores diferentes:

- *Control Point* (Punto de Control): se trata del *Control Point* de la arquitectura UPnP para audio y video. Permite al usuario controlar al MediarRenderer además de recibir la información de estado de los servicios asociados a éste.

- Plataforma OSGi: a través de la implementación de la plataforma OSGi se controla el ciclo de vida de la aplicación al estar integrada en un *bundle*, permitiendo que el dispositivo se inicie y agregue a la red UPnP o que la abandone al parar su ejecución.

- *UPnP MediaServer*: se trata de los dispositivos *MediaServer*s de UPnP disponibles en la red. Proporcionan el contenido multimedia que se comparte en la red UPnP para poder ser renderizado por el *MediaRenderer*.

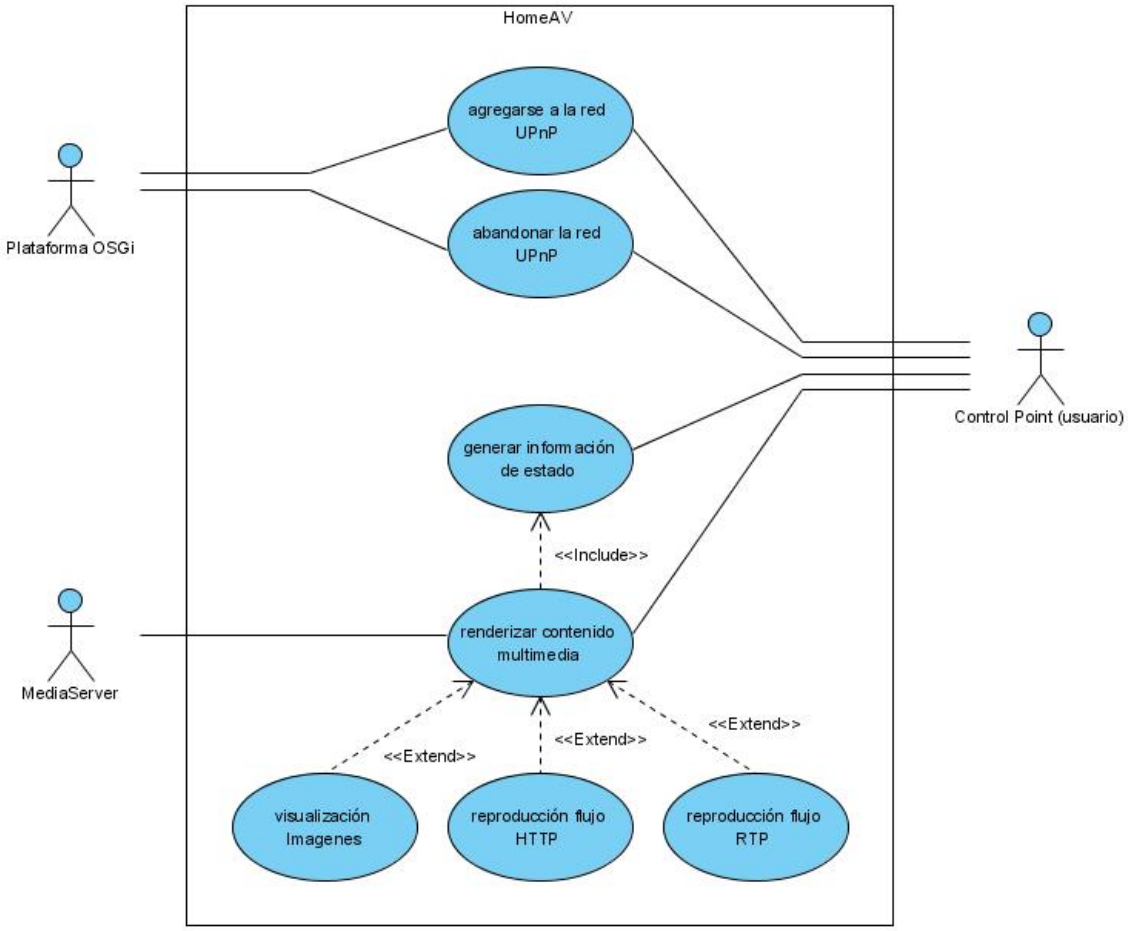


Figura 3.1. Diagrama de Casos de Uso

A continuación se definen brevemente los casos de uso del diagrama:

1. Agregarse a la red UPnP

- Descripción: La plataforma OSGi controla el ciclo de vida del *bundle*, y por consiguiente de la aplicación. Por tanto, cuando el *bundle* pasa a estado *Active*, inicia el proceso para agregarse a la red UPnP y ser descubierto por el *Control Point*.
- Actores implicados: Plataforma OSGi, *Control Point*.
- Casos de Uso relacionados:-
- Precondiciones: -
- Postcondiciones: tras el descubrimiento del dispositivo, el *Control Point* puede controlar la renderización del contenido y recibir información del estado de los servicios asociados a éste.

2. Abandonar la red UPnP

- Descripción: Cuando se para el *bundle* y va a pasar al estado *Resolved* de su ciclo de vida, la aplicación inicia el proceso para parar todos los hilos de ejecución relativos a la renderización y abandonar de forma ordenada la red UPnP.
- Actores implicados: Plataforma OSGi, *Control Point*.
- Casos de Uso relacionados:-
- Precondiciones: Es necesario que el dispositivo se haya agregado con éxito a la red.
- Postcondiciones: La aplicación deja de estar disponible.

3. Generar información de estado

- Descripción: UPnP dispone de un mecanismo de eventos para el envío de información al *Control Point* sobre el estado de los servicios asociados al dispositivo. La aplicación genera y envía la información necesaria cuando y como la especificación UPnP establece. Un proceso de renderización incluye el modelado del estado de los servicios relacionados con este proceso (*AVTransport Service* y *RenderingControl Service*) generando la correspondiente información.
- Actores implicados: *Control Point*
- Casos de Uso relacionados: Renderizar contenido multimedia
- Precondiciones: Es necesario que el dispositivo se haya agregado con éxito a la red.
- Postcondiciones:-

4. Renderizar contenido multimedia

- Descripción: El usuario, a través del *Control Point*, controla la renderización del contenido multimedia localizado por el *MediaServer*. La aplicación procesará de forma distinta este contenido en función del tipo que sea: una imagen, o un flujo HTTP de audio y/o video, o un *streaming* RTP.
- Actores implicados: *MediaServer*, *ControlPoint*.
- Casos de Uso relacionados: Visualización imágenes, reproducción flujo HTTP, reproducción flujo RTP, generar información de estado.
- Precondiciones: Es necesario que el dispositivo se haya agregado con éxito a la red.
- Postcondiciones:-

En la siguiente tabla se muestra la relación entre los requisitos del sistema y los casos de uso descritos:

Casos de Uso / Requisitos	**Caso 1** Agregarse a la red UPnP	**Caso 2** Abandonar la red UPnP	**Caso 3** Generar información de estado	**Caso 4** Renderizar contenido multimedia
Compatibilidad con UPnP	X	X	X	X
Control de la reproducción			X	X
Visualización de imágenes				X
Reproducción de flujos RTP				X
Reproducción de flujos HTTP				X
Compatibilidad con IPv6	X	X	X	X
Multiplataforma	X	X	X	X
Integración en OSGi	X	X		
Reproducción de formatos comunes				X
Extensibilidad			X	X

Tabla 3.1. Casos de Uso frente a requisitos

3.2 Herramientas utilizadas

Uno de los primeros pasos del desarrollo de un proyecto es definir el entorno de desarrollo en el que se va a trabajar. A continuación se enumeran las herramientas que han sido utilizadas durante todo el proceso de creación de la aplicación.

Sistemas Operativos

Los sistemas operativos que se han utilizado para el desarrollo de la aplicación han sido, Windows XP en su versión *Home Edition* y una distribución GNU/Linux, la Ubuntu versión 8.04. En cada fase de la implementación el código se ha probado en ambas plataformas.

Programación

La utilización de Java como lenguaje de programación ha permitido cumplir varios de los requisitos del sistema descritos con anterioridad. Java es un lenguaje multiplataforma, es decir, que el código puede ser ejecutado en todas las plataformas que dispongan de Máquina Virtual Java. Por otra parte, Java soporta el direccionamiento con IPv6, y el desarrollo de interfaces gráficas y manejo de imágenes a través de paquetes integrados en el API para varias versiones de la plataforma J2SE (Java Swing, AWT). En este proyecto se han utilizado las versiones JDK1.6 para Windows y JDK1.7 para Ubuntu.

Como ya se ha comentado cuando se presentó JMF, ésta librería multimedia proporciona a la aplicación el procesado de datos multimedia desde su obtención a su renderización. Permite el control de la reproducción por medio de una sencilla invocación de métodos que definen acciones comunes de un reproductor, como las mencionadas en los requisitos y otras más. También se dijo que JMF posee un API para RTP con el que dar soporte a la implementación para abrir sesiones RTP y recibir flujos en este formato.

En el apartado “Estado del Arte” del capítulo “Marco teórico”, se habló sobre las limitaciones de JMF respecto de los formatos soportados. Para aumentar el número de formatos soportados se han utilizado los siguientes *plug-ins*:

- Fobs4JMF: para formatos de video como, los contenidos en un fichero AVI (divx, xvid,...), h264, etc.
- Jffmpeg: para la decodificación MP3 (MPEG-1 *Layer* 3) en contenidos con audio y video.
- mp3plug-in: para la decodificación MP3 de archivos de audio.

Por otra parte, para incorporar toda la funcionalidad sobre protocolos UPnP, y por tanto cumplir con la especificación, se ha utilizado el conjunto de clases Java que provee la aplicación Cidero [CIDERO]. Se hablará detalladamente sobre su uso en el apartado “Implementación de la aplicación: HomeAV”.

Entorno de Desarrollo Integrado

Para la programación se ha de utilizar un software que integre un conjunto de herramientas para el desarrollo de programas en Java y que facilite esta tarea (IDE, *Integrated Development Environment*). En este caso se ha utilizado un software de código abierto denominado Eclipse. La versión utilizada es la 3.3.2. [ECLIPSE]

Plataforma OSGi

Uno de los requisitos no funcionales del sistema es su integración en una plataforma OSGi. La implementación del *Realease* 4 del *Framework* de OSGi que se ha utilizado es la que provee la versión mencionada de *Eclipse* y que lleva el nombre de *Equinox* en su versión 3.3.0. Aunque para la realización de pruebas también se ha utilizado la versión 1.8.0 de *Felix*.

Dispositivos UPnP

Para crear el escenario definido por la arquitectura AV de UPnP se han utilizado los siguientes dispositivos que cumplen con el estándar:

- *Control Point* → la versión 1.5.3 de Cidero. Se trata de una aplicación de código abierto cuya librería está basada en la de Cyberlink [CYBERGARAGE] para la creación de dispositivos UPnP.

- *UPnP MediaServer* → se han utilizado dos aplicaciones: CMGate [CYBERGARAGE], aplicación de código abierto y basada en Cyberlink, y la versión 11 del Windows Media Player [WMP11].

Servidores de streaming RTP

En el momento en el que se realizó este proyecto todos los *UPnP MediaServer* que se encontraron facilitaban el acceso a los contenidos multimedia a través del protocolo HTTP. En ningún caso tenían capacidad para realizar *streaming* RTP. Por ello, se utilizaron aplicaciones no compatibles con UPnP para el envío de flujos RTP (se hace referencia a esto en el capítulo “Pruebas”). Estos programas son:

- JMStudio, aplicación propiedad de Sun Microsystems que se distribuye junto con la librería JMF.

- RTPTransmite3, clase Java desarrollada por Sun Microsystems. Se trata de un pequeño programa, basado también en la librería JMF, y que permite transmitir y recibir flujos RTP utilizando *sockets* UDP [JMFCUSTOMRTP].

Otras herramientas

- Ethereal: es un analizador de protocolos. Se ha utilizado para el análisis y monitorización de las comunicaciones de red entre los dispositivos UPnP.

- Herramientas de edición de diagramas UML: MaintainJ [MAINTAINJ] en su versión 2.5.3, para generar los diagramas de secuencia, y SDE (*Smart Development Environment*) [SDE] en su versión 4.0, para los diagramas de caso de uso y de clase.

3.3 Implementación de la aplicación: HomeAV

En este apartado se presenta la aplicación que ha sido desarrollada. Se hará una breve introducción en la que se describirá el proceso de su implementación. Posteriomente se enunciarán las características funcionales principales de la misma, seguido de una síntesis de su arquitectura de clases y métodos. Finalmente, se muestran los diagramas de secuencia de los procesos de mayor relevancia entre todos los que suceden durante su ejecución, y un último esquema representando los distintos estados en los que se puede encontrar el proceso de renderización.

3.3.1 Introducción

En las próximas líneas se describen las fases que han compuesto el proceso de implementación de la aplicación, cada una de las cuales representa una unidad funcional de la misma. Estas son:

1. Creación de un dispositivo UPnP y definición de los servicios asociados a un *MediaRenderer*.

El paso previo a la programación del dispositivo es la creación de los documentos de texto en formato XML. Uno como descriptor del dispositivo y los respectivos descriptores de los servicios asociados al mismo.

Para la programación, en esta fase se ha utilizado la librería Cyberlink [CYBERGARAGE] que se distribuye con la aplicación Cidero. Esta proporciona toda la funcionalidad necesaria para la creación de un dispositivo UPnP. Implementa cada uno de los protocolos que especifica el estándar (SSDP, SOAP,…), y por consiguiente, cada una de las fases en la comunicación UPnP entre dispositivo y *Control Point*. Para ello define un conjunto de métodos y clases perfectamente estructurado. (Ver “ANEXO I” sobre Cidero/Cyberlink)

Para la implementación de los servicios asociados a un *UPnP MediaRenderer* también se ha utilizado la librería de Cidero. Esta librería, a parte de contener la ya mencionada Cyberlink, posee un catálogo de métodos y clases propios que definen perfectamente los servicios que componen un *UPnP MediaRenderer*, y sus respectivas acciones y variables de estado. Se implementaron los servicios: *ConnectionManager Service*, *AVTransport Service* y *RenderingControl Service*. (Ver “ANEXO I” sobre Cidero/Cyberlink)

También en esta fase se define la interfaz gráfica mínima de la aplicación utilizando el paquete Java Swing que ya mencionamos en el apartado “Herramientas utilizadas”. Esta interfaz no dispone de botonera, y únicamente servirá para poder añadir el componente visual del video que se vaya a reproducir y algunos parámetros básicos asociados a la reproducción como son el título del medio, la duración y la referencia temporal de la reproducción.

2. Reproducción de contenido multimedia (HTTP)

Una vez conocido y adaptado el código necesario para la comunicación con el *Control Point*, incluyendo el conjunto de acciones de los servicios que pueden ser invocadas, las variables de estado, y toda la lógica asociada a esto, se procedió a la fase para la implementación de las capacidades de reproducción de contenido multimedia. Para ello, como ya se ha comentado en varias ocasiones, se ha utilizado el API de JMF. En esta fase se estudió e implementó el código necesario para la reproducción de contenidos multimedia, audio y video,

alojado en otro equipo y obtenido a través de la red utilizando el protocolo HTTP. Para ello, se utilizaron los métodos, clases e interfaces, de entre las que proporciona JMF, para la creación del reproductor multimedia (`Player`), la invocación de sus controles básicos (`Play()`, `Stop()`, `Pause()`, `SetVolume()`, `SetMute()`,...) asociándolos con las acciones definidas para ello por los servicios UPnP implementados, *AVTransport Service* y *RenderingControl Service*.

También se crea un mecanismo, basado en generación y escucha de eventos, para trasladar información acerca del éxito de la invocación de los controles mencionados y del estado en el que se encuentra el *Player* a las clases que implementan los servicios UPnP anteriores, y que necesiten esta información para definir su modelo de estado e informar de las modificaciones en éste al *Control Point*.

3. Visualización de imágenes

En esta fase se añade la capacidad de visualizar imágenes. Para ello se hace uso de los ya mencionados paquetes, Java Swing y AWT.

Además, se incorpora la funcionalidad necesaria al mecanismo para el traslado de información a los servicios UPnP que se derive del proceso de visualización de imágenes.

4. Recepción y reproducción de flujos RTP

Para la recepción y reproducción de flujos RTP se vuelve a hacer uso de JMF, en concreto de la especificación de la misma para *streaming* RTP. En este caso se implementó un receptor de *streaming* RTP que crea sesiones RTP para la recepción, utilizando *sockets* UDP. Para conseguir esto, se han utilizado las clases de un pequeño programa creado por Sun Microsystems, que también se utiliza en las pruebas para la transmisión de flujo RTP, y que ya se presentó en el apartado “Herramientas del Sistema” en la sección que habla sobre servidores de *streaming* RTP.

Las acciones que definen los servicios, al igual que con el flujo HTTP, son asociadas con la invocación de los métodos para la creación de la sesión RTP (`Play()`), pausar la transmisión (`Pause()`), cerrar la sesión (`Stop()`), o modificar el volumen (`SetVolume()` o `SetMute()`).

Al igual que en la fase anterior, se añade la funcionalidad necesaria el mecanismo de generación y escucha de eventos para trasladar la información relativa al estado de la sesión RTP o informar del éxito de las invocaciones a los métodos relacionados con el control de esta o la renderización.

5. Integración en OSGi

Una vez creado el *UPnP MediaRenderer* con la funcionalidad requerida y habiendo sido probado en un escenario UPnP real, se procedió a transformar la aplicación en un *bundle* OSGi para poder ser integrada en una plataforma OSGi. Para ello, se utilizó el API de OSGi en la versión para el *Release* 4 de la plataforma.

En cada una de estas fases, el código generado era sometido a un proceso de depuración con el que se comprobaba el correcto funcionamiento del sistema. Posteriormente era probado tanto en un equipo Windows como en un equipo Linux, y utilizando tanto el protocolo IPv4 como IPv6 en las comunicaciones. Con la realización de estas pruebas se aseguraba la independencia del código respecto de ambas plataformas y la compatibilidad de la aplicación con ambos protocolos de red.

3.3.2 HomeAV

Figura 3.2. Interfaz Gráfica de HomeAV

En las próximas líneas se presenta el prototipo de *UPnP MediaRenderer* que se ha diseñado.

Se ha pretendido implementar una estructura lógica lo más sencilla e intuitiva posible que facilite no sólo su compresión sino también la integración de mayores capacidades en un futuro.

Se ha utilizado el Inglés en toda la programación y para comentar el código. Esto viene motivado principalmente por el carácter global de las tecnologías en las que se basa la aplicación, así como por guardar una mayor coherencia con la utilización de librerías Java definidas en este idioma.

Se han declarado algunos métodos como estáticos cuando se ha considerado que se facilitaba su uso, no teniendo que instanciar las respectivas clases que los contienen para ser invocados.

¿Qué es HomeAV?

HomeAV es el nombre con el que se ha bautizado el prototipo. Se trata de una aplicación que conforma un dispositivo *UPnP MediaRenderer* con las siguientes capacidades:

- Compatibilidad con UPnP, dando soporte a todos los protocolos definidos por el estándar y permitiendo la comunicación con los *AV Control Point* de UPnP que se encuentren en la red doméstica.

- Da soporte a la acción *SetNextAVTrasnportURI()* del servicio AVTransport de un *UPnP MediaRenderer* para una reproducción continua de ítems, es decir, sin huecos temporales entre ítem e ítem.

- Reproduce el contenido multimedia disponible en la red UPnP doméstica, obtenido utilizando el protocolo de transporte HTTP. Este contenido incluye audio y video con los formatos más comunes (MPEG, DIVX, AVI, MP3, WAV,...).

- Permite el almacenamiento temporal en local (*caching*) del contenido multimedia previamente a su renderización. Únicamente tiene sentido para el contenido, ficheros de audio y video, obtenido mediante HTTP.

- Visualiza imágenes con los formatos más comunes (JPEG, GIF, PNG, TIFF).

- Reproduce el contenido de audio y video disponible en la red UPnP por medio de flujos RTP, pudiendo abrir dos sesiones para flujos de audio y video respectivamente enviados a la misma IP pero a puertos contiguos. Soporta los formatos RTP más comunes de audio y video (MPEG, JPEG, H623, MPEGAudio, G.723, GSM, DVI,...).

- Soporta el protocolo IP de nueva generación, IPv6.

- Integración en una plataforma OSGi. Un *bundle* OSGi contendrá a la aplicación para poder ser desplegada en pasarelas domésticas.

3.3.3 Arquitectura de clases

En este apartado se describen el conjunto de paquetes y clases que componen HomeAV. También, por su importancia, se mencionan algunos métodos definidos en ellas.

En la siguiente figura se muestra el diagrama de clases de HomeAV. Se recomienda consultar el “ANEXO III” sobre Diagramas UML para su mejor interpretación.

Posteriormente, se describen los tres módulos funcionales en los que se ha subdividido la aplicación: Módulo raíz, Módulo UPnP y Módulo multimedia. Cada uno de ellos corresponde con cada uno de los paquetes principales de clases e interfaces Java que aparecen en el diagrama.

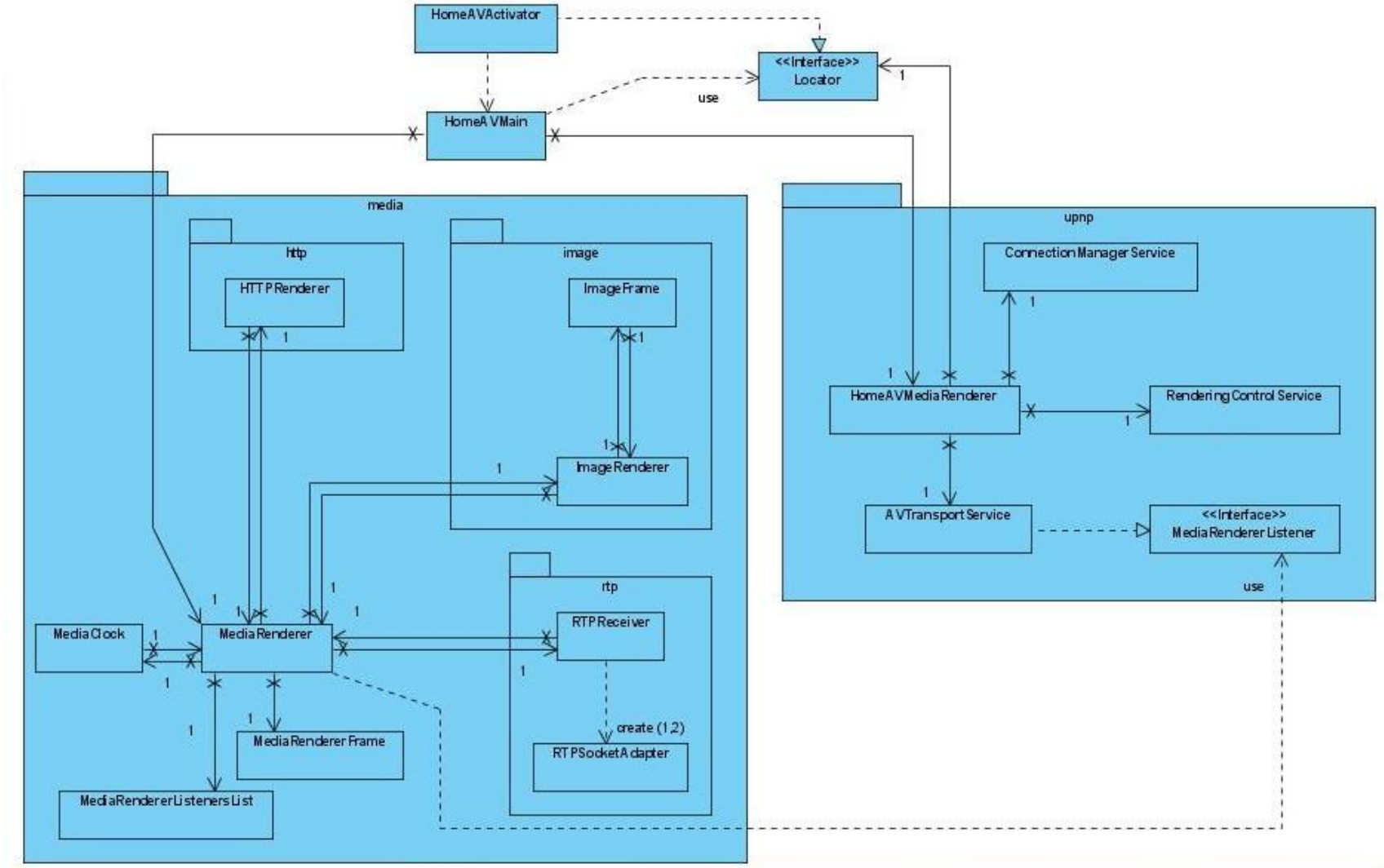


Figura 3.3. Diagrama de Clases de HomeAV

Módulo raíz

Se trata del módulo principal. A parte de contener los paquetes de clases `homeAV.upnp` y `homeAV.media` para implementar funcionalidades concretas del sistema, es el que gestiona la puesta en marcha y el cierre de la aplicación.

Paquete	Clases contenidas	Interfaces contenidas
`homeAV`	`HomeAVActivator` `HomeAVMain`	*`Locator`*
Paquetes contenidos		
`homeAV.upnp` `homeAV.media`	*<<Ver los respectivos módulos>>*	

Tabla 3.2. Composición del Módulo raíz

Clases

- `HomeAVActivator` → es la clase que implementa a la interfaz *`BundleActivator`* que provee el API de OSGi. Define los métodos para provocar el arranque (`start()`) y la parada (`stop()`) del *bundle*, los cuales serán usados por la implementación OSGi para la gestión del ciclo de vida del *bundle*.

  Implementa la interfaz *`Locator`* para declararse como localizador y definir los métodos para la obtención de ficheros que se encuentren dentro del directorio de la aplicación, es decir, en el comprimido (.jar) del *bundle* y a los que se accede por medio del contexto de ejecución del mismo. Al llamar al método de arranque de `HomeAVMain` (`start()`), se pasa a sí mismo como argumento. De esta forma `HomeAVMain` puede hacer uso de estos métodos.

- `HomeAVMain` → es la clase principal de HomeAV. Es utilizada por la anterior clase para arrancar y cerrar la aplicación por medio de la invocación de los respectivos métodos `start()` y `destroy()`.

  El primer paso en el arranque es la carga de los ficheros que se necesitan durante el inicio. Para ello hace uso de la instancia de la clase que implementa *`Locator`*, en este caso `HomeAVActivator`. Estos ficheros son:

  - *homeAV.config*: documento en texto plano con el que se configura los parámetros de inicio de la aplicación. (Ver "ANEXO IV: Ficheros de la aplicación")
  - *descriptor.xml*: fichero en formato XML con la descripción UPnP del dispositivo (aplicación). Este fichero será utilizado cuando se inicie la comunicación UPnP. (Ver "ANEXO IV: Ficheros de la aplicación")

  Se crea una instancia estática de cada una de las clases principales de cada paquete que contiene, `MediaRenderer` (`homeAV.media`) y `HomeAVMediaRenderer` (`home.upnp`), como consecuencia de su uso en un método estático. Y además, en el primer caso, facilita la invocación de sus métodos desde otros métodos de otras clases que no pertenecen a su mismo paquete.

- *`Locator`* (interfaz): es utilizada para definir localizadores que permitan obtener los ficheros que se encuentren dentro del directorio de la aplicación, es decir, en el comprimido (.jar) del *bundle*.

Módulo UPnP

Este módulo asume toda la funcionalidad relacionada con la implementación del protocolo UPnP definida para un *MediaRenderer*.

Paquete	Clases contenidas	Interfaces contenidas
homeAV.upnp	HomeAVMediaRenderer AVTransportService ConnectionManagerService RenderingControlService	*MediaRendererListener*
Paquetes contenidos		

Tabla 3.3. Composición del Módulo UPnP

Clases

- HomeAVMediaRenderer: es la clase principal del paquete homeAV.upnp y la que extiende la clase Device de la librería Cyberlink. Se encarga de la instanciación de los servicios UPnP del dispositivo, AVTransportService, ConnectionManagerService y RenderingControlService. Proporciona los métodos necesarios para la creación del UPnP Device y el comienzo (start()) y parada (stop()) de la comunicación UPnP.

  En su constructor, inicializa el dispositivo utilizando el descriptor del mismo y estableciendo los parámetros de configuración UPnP iniciales como el puerto HTTP a utilizar, la dirección IP a utilizar en la comunicación y el FriendlyName[1] del dispositivo con los dos últimos octetos de esta IP (FriendlyName="HomeAV (IP:)")

  Uno de los pasos más importantes que realiza esta clase es el de añadir la instancia de AVTransportService a la lista de escuchadores de la clase MediaRenderer, lo que permitirá informar la servicio del estado en la reproducción o visualización de imágenes y modelar en consecuencia su estado.

- *MediaRendererListener* (interfaz): es utilizada para definir escuchadores de los eventos que genere la clase MediaRenderer del paquete homeAV.media.

- AVTransportService: implementa la clase AVTransport del paquete com.cidero.upnp de Cidero. Por tanto, se trata de la implementación del servicio *AVTransport Service* de UPnP. Esta clase proporciona el conjunto de todos los métodos asociados a las acciones de este servicio y que el *Control Point* invoca para el control del flujo, la reproducción o la visualización del contenido multimedia.

  Como resultado de la invocación de una acción, actualiza las variables de estado del servicio relacionadas con dicha acción. Además, implementa la interfaz *MediaRendererListener*, e implementa el método que define ésta, mediaRendererEventReceived(), para poder enterarse de los eventos generados por los cambios producidos en el tratamiento del contenido multimedia asociado y actualizar en consecuencia las variables de estado que sean necesarias. A continuación se muéstra el código de este método.

[1] FriendlyName: parámetro UPnP que contiene el nombre del dispositivo con el que se va a publicar en la red.

```
	/**
	 * This method is called when an event is generated by the
	 * 'MediaRenderer' instance that this listener is registered
	 * with.
	 */
	@Override
	public void mediaRendererEventReceived(String type) {
		// TODO Auto-generated method stub
		if(type.equals("StartEvent")){

	if(!getStateVariableValue("TransportState").equals(AVTransport.STATE_PLAY
ING))
		updateStateVariable("TransportState", AVTransport.STATE_PLAYING );
		}
		else if(type.equals("EndOfMediaEvent")){
			if(HomeAVMain.renderer.nextPlayback){
				setNextPlayback();
			}else{
				logger.info("--- EndOfMediaEvent ---. No next
				playback");
				uri = null;
				uriMetaData = null;
				//Update the state variables
				setURIRelatedStateVariables(INIT_URI,
				INIT_URIMETADATA, INIT_TIME);
				updateStateVariable("TransportState",
				AVTransport.STATE_STOPPED );
			}
		}
		else if(type.equals("ControllerErrorEventFromPlayer")){
			uri = null;
			uriMetaData = null;
			if(nextURI != null)
				nextURI = null;
			if(nextURIMetaData != null)
				nextURIMetaData = null;
			//Update the state variables
			setURIRelatedStateVariables(INIT_URI, INIT_URIMETADATA,
			INIT_TIME);
			updateStateVariable("TransportState",
			AVTransport.STATE_STOPPED );
		}
		else if(type.equals("ControllerErrorEventFromPlayerNextURI")){
			nextURI = null;
			nextURIMetaData = null;
		}
		else if(type.equals("SeekFailedEvent")){
			uri = null;
			uriMetaData = null;
			if(nextURI != null)
				nextURI = null;
			if(nextURIMetaData != null)
				nextURIMetaData = null;
			//Update the state variables
			setURIRelatedStateVariables(INIT_URI, INIT_URIMETADATA,
			INIT_TIME);
			updateStateVariable("TransportState",
			AVTransport.STATE_STOPPED );
		}
```

```
            else if(type.equals("ControllerErrorEventFromRTPPlayer")){
                  uri = null;
                  uriMetaData = null;
                  //Update the state variables
                  setURIRelatedStateVariables(INIT_URI, INIT_URIMETADATA,
                  INIT_TIME);
                  updateStateVariable("TransportState",
                  AVTransport.STATE_STOPPED );
            }
            else if(type.equals("ByeEvent")){
                  logger.info("--- ByeEvent ---");
                  uri = null;
                  uriMetaData = null;
                  //Update the state variables
                  setURIRelatedStateVariables(INIT_URI, INIT_URIMETADATA,
                  INIT_TIME);
                  updateStateVariable("TransportState",
                  AVTransport.STATE_STOPPED );
            }
            else if(type.equals("ImageClosedEvent")){
            if(HomeAVMain.renderer.nextPlayback){
      setNextPlayback();
                  }else{
                        logger.info("--- ImageCloseEvent ---");
                        uri = null;
                        uriMetaData = null;
                        //Update the state variables
                        setURIRelatedStateVariables(INIT_URI,
                        INIT_URIMETADATA, "-");
                        updateStateVariable("TransportState",
                        AVTransport.STATE_STOPPED );
                  }
            }
      }
```

Cuadro de texto 3.1. Método `mediaRendererEventReceived()` de `AVTransportService`

Las acciones relacionadas con la reproducción que están implementadas son las usuales del *AVTransport Service*: `Play()`, `Stop()`, `Pause()`, `Seek()` y `SetPlayMode()` para definir el modo de reproducción (*NORMAL*, *REPEAT_ONE* o *REPEAT_ALL*). También se da soporte para la invocación de la acción `Seek()`, para buscar un punto temporal en la reproducción de un fichero multimedia obtenido mediante HTTP. Las acciones `Next()` y `Previous()` no producen ningún efecto en la reproducción en HomeAV (en el capítulo "Conclusiones" se justificará el motivo). Los métodos asociados a cada acción de este tipo, invocan a los respectivos métodos de la clase `MediaRenderer` del paquete `homeAV.media` donde se procesa el contenido. Para esas invocaciones no es necesario haber creado una instancia de la clase `MediaRenderer` ya que, como se dijo en la descripción del Módulo raíz, se utiliza la que se creó en `HomeAVMain` declarada como estática.

La acción `SetAVTransportURI()` es la primera en invocarse para el comienzo de una nueva renderización. Establece la URI del recurso multimedia que va a ser renderizado, y en la cual se indica el protocolo de transporte con el que se va a transferir el contenido. Su invocación va seguida de la invocación de la acción `Play()`. HomeAV utiliza la primera para la creación del `Player` en el caso de flujos HTTP de audio y/o video, o para la creación de la sesión RTP en el caso de que se trate de una URL para un flujo RTP. La segunda es utilizada para arrancar el `Player`, o para crear el display de una imagen, o para iniciar la sesión RTP, según proceda.

Cuando se presentó HomeAV en páginas anteriores, se indicó que una de sus cualidades es soportar la acción `SetNextAVTransportURI()` de este servicio. Esta acción es

invocada por el *Control Point* durante la renderización del contenido actual, precisamente para que cuando este finalice pueda inmediatamente empezar la renderización del siguiente. Ésta acción es realmente útil para aquellos recursos que necesiten ser almacenados en local previamente a su renderización, en este caso los que se obtienen a través de la red mediante HTTP. El método de `AVTransportService` asociado a esta acción invoca al método de `MediaRenderer`, `prepareNextPlayback()`, encargado de preparar la próxima reproducción.

- `ConnectionManagerService`: implementa a la clase `ConnectionManager` del paquete `com.cidero.upnp` de Cidero. Por tanto, se trata de la implementación del servicio *ConnectionManager Service* de UPnP. Implementa métodos para las acciones que el *Control Point* invoca para la obtención de información sobre los protocolos y formatos soportados por el *UPnP MediaRenderer*, y la información sobre los identificadores de los servicios y de la conexión.

  Una de las decisiones que se tomó fue la de no implementar la lógica de las acciones del servicio, `PrepareForConnection()` y `ConnectionComplete()` (en el capítulo "Conclusiones" se justificará el motivo). El resto de acciones del servicio sí son implementadas.

- `RenderingControlService`: implementa a la clase `RenderingControl` del paquete `com.cidero.upnp` de Cidero. Por tanto, se trata de la implementación del servicio *RenderingControl Service* de UPnP. Implementa métodos para las acciones que el *Control Point* invoca para el manejo de algunas características de la renderización. En concreto, las que son obligatorias y las asociadas al control del volumen y el mute. Al igual que en la clase `AVTransportService`, estos métodos asociados a cada acción de este tipo, invocan a los respectivos métodos de la clase `MediaRenderer` del paquete `homeAV.media` para actuar sobre estas características del reproductor.

Módulo multimedia

Este módulo asume toda la funcionalidad relacionada con el tratamiento de contenido multimedia para su reproducción o visualización. Contiene los paquetes de especificación `homeAV.media.http`, `homeAV.media.image` y `homeAV.media.rtp`.

Paquete	**Clases contenidas**	**Interfaces cont.**

`homeAV.media`	`MediaRenderer`
	`MediaRendererFrame`
	`MediaClock`
	`MediaRendererListenerList`	

Paquetes contenidos		
`homeAV.media.http`	`HTTPRenderer`	

`homeAV.media.image`	`ImageFrame`
	`ImageFrame.ImageLabel`
	`ImageRenderer`	

`homeAV.media.rtp`	`RTPReceiver`
	`RTPReceiver.Instance`
	`RTPReceiver.SessionLabel`
	`RTPSocketAdapter`
	`RTPSocketAdapter.SockInputStream`
	`RTPSocketAdapter.SockOutputStream`	

Tabla 3.4. Composición del Módulo multimedia

Clases

- `MediaRenderer`: es la clase principal del paquete `homeAV.media`. Controla la renderización del contenido multimedia. Clasifica el contenido multimedia a renderizar a fin de invocar los métodos adecuados de los manejadores de la renderización específicos

de cada tipo (Audio/Video HTTP, Imagen, Flujo RTP). Para ello, crea las instancias de los estos manejadores (`HTTPRenderer`, `ImageRenderer` y `RTPReceiver`). Con esto se pretende que cuando un método de la clase `AVTransportService`, relativo a una acción, llame al método de esta clase que la implementa sobre la renderización, éste invoque al método correspondiente de la subclase manejadora que procesa el tipo concreto de contenido asociado. Sirva como ejemplo de ello el método `play()` de esta clase que se muestra a continuación.

```
/**
 * Start the playback.
 * @return uriFor or FAILURE if no success
 */
public int play(){
	if(uriFor == HTTP_AV_URI){
		if(startAtSpecificTime){
			if(httpRenderer.setTimePosition( startAtSpecificTime,
			startTime))
				logger.info("Starting player at specific
				time");
			startAtSpecificTime = false;
			startTime = null;
		}
		if(httpRenderer.realizePlayer())
			return uriFor;
		else
			return FAILURE;
	}else if(uriFor == HTTP_IMAGE_URI){
		if(startAtSpecificTime){
			startAtSpecificTime = false;
			startTime = null;
			logger.warning("Impossible to start at specific time.
			The uri is for an image");
		}
		if(imageRenderer.createImageDisplay(uri, getTitleString()))
			return uriFor;
		else
			return FAILURE;
	}else{
		if(startAtSpecificTime){
			startAtSpecificTime = false;
			startTime = null;
			logger.warning("Impossible to start at specific time a
			RTP session");
		}
		if(rtpReceiver.initializeRTPSession())
			return uriFor;
		else
			return FAILURE;
	}
}
```

Cuadro de texto 3.2. Método `play()` de `MediaRenderer`

De acuerdo con la lógica descrita en la clase `AVTransportService`, este método `play()` se invoca en el método `actionPlay()` de esa clase haciendo que el `Player` que había sido creado antes pase a estado *Realize*, o creando el display para visualizar una imagen, o iniciando la sesión RTP, según sea el tipo de contenido asociado que se pretende renderizar.

Para dar soporte a la invocación de la acción `SetNextAVTrasnportURI()`, como se dijo, esta clase provee el método `prepareNextPlayback()`. Este método, conforme a lo dicho anteriormente, identifica el tipo de contenido que se va a renderizar permitiendo

resolver con éxito esta invocación únicamente si se trata de audio o video obtenido mediante HTTP.

Otra de las funcionalidades que asume esta clase es la de recoger los eventos generados por las clases manejadoras. Para ello, provee tres métodos, uno por cada tipo de manejador, que son invocados por estos cuando se genera un evento en la renderización, ya sea del `Player` en ejecución, o respecto de la sesión RTP, o por el cierre de la ventana de visualización de la imagen, en función del manejador que se esté utilizando para esa renderización. Estos métodos son `httpRendererEventReceived()`, `imageRendererEventReceived()` y `rtpReceiverEventReceived()`.

Posteriormente, notifica estos eventos a los escuchadores registrados de la clase, en este caso `AVTransportService`, por medio de la invocación del método de estos escuchadores `notifyRendererEvent()`.

A partir de todo lo anterior, se puede decir, que `MediaRenderer`, hace las funciones de filtro entre las instrucciones de la clase `AVTransportService` y las subclases manejadoras de tipos concretos de contenidos. Y, a su vez, permite que todos los posibles eventos generados por estas subclases puedan ser trasladados de forma coherente a la misma clase, `AVTransportService`.

- `MediaRendererFrame`: proporciona la interfaz gráfica de HomeAV. Consiste únicamente en una ventana que incluye dos etiquetas, una para añadir el título del medio y otra para añadir la referencia temporal de la reproducción y la duración. Además, si el contenido asociado tiene componente visual, éste se añadirá también a la ventana principal.

- `MediaClock`: esta clase es utilizada para crear la referencia temporal de la reproducción obtenida del `Player` (si existe) que esté en ejecución. Y crea la cadena de texto con esta referencia y la duración que se añadirá a la ventana principal.

- `MediaRendererListenerList`: esta clase extiende de `Vector` y es usada por la clase `MediaRenderer` para la crear una lista de los escuchadores de eventos generados por ella.

Paquetes contenidos

1. `homeAV.media.http`: contiene el conjunto de métodos y clases para la reproducción de audio y video obtenido de la red utilizando HTTP como protocolo de transporte.

   Clases:

- `HTTPRenderer`: provee el conjunto de métodos necesarios para la creación y control de una instancia de la clase `Player` de la librería JMF para el contenido que es obtenido por HTTP. Entre los controles disponibles están aquellos que permiten la variación del volumen y el mute mediante la invocación de métodos. Implementa la interfaz escuchadora *`ControllerListener`* para poder enterarse de los eventos producidos por el `Player` durante su existencia y actuar en consecuencia. Esto incluye la invocación de uno de sus métodos para la notificación de estos eventos a la clase `MediaRenderer` y que ésta pueda así difundirlos, de ser necesario, a sus escuchadores de eventos (`AVTransportService`).

  El método más importante de esta clase es `createPlayer()`. Este método es usado para la creación del `Player`, ya sea para preparar la reproducción del contenido entrante en respuesta a la invocación de la acción `SetAVTransportURI()` o para preparar la del próximo recurso en respuesta a la acción `SetNextAVTransportURI()`. En el primer caso, simplemente se crea el `Player` que posteriormente se “realizará” (pasará al estado

Realice) cuando el *Control Point* invoque la acción `Play()` del servicio *`AVTransport Service`*. En el segundo caso, se crea un `Player` en estado *Realize*, estando el actual `Player` en ejecución, con lo que el contenido del siguiente recurso ya se habrá descargado, y el nuevo `Player` arrancará cuando el actual llegue al final de la reproducción y se cierre.

```
/**
 * Create a Player instance from a specific URL.
 * @param uri
 * @param isForNextPlayback 'true'->create a Player for next resource
 * @return   'true' if success
 */
public boolean createPlayer(String uri, boolean isForNextPlayback){
	DataSource ds = null;
	Player localPlayer = null;
	MediaLocator ml = new MediaLocator(uri);
	if(ml != null){
		try{
			localPlayer = Manager.createPlayer(ml);
			localPlayer.addControllerListener(this);
			if(isForNextPlayback){
				playerForNextURI = localPlayer;
				localPlayer = null;
				playerForNextURI.realize();
				return true;
			}else{
				player = localPlayer;
				localPlayer = null;
				return true;
			}
		}
		catch (Exception e){
			logger.warning("An error occurs creating the player");
			e.printStackTrace();
			localPlayer.close();
			localPlayer = null;
			return false;
		}
	}else{
		logger.warning("There is no player for this URI");
		return false;
	}
}
```

Cuadro de texto 3.3. Método `createPlayer()` de la clase `HTTPRenderer`

La reproducción para cuando se llega al final del medio o por la invocación de la acción `Stop()` del servicio, por parte del *Control Point*. Esta invocación supone la llamada al método `stop()` de `MediaRenderer` que provoca, no sólo lo dicho anteriormente sino también la cancelación y cierre del proceso que prepara la siguiente reproducción.

2. `homeAV.media.image`: contiene el conjunto de métodos y clases para la visualización de imágenes.

   Clases:

- `ImageRenderer`: es la clase manejadora de imágenes. Contiene el conjunto de métodos necesarios para crear, actualizar y cerrar el *display* (ventana) donde se visualizará la imagen. Como la clase anteriormente vista, incluye un método para notificar un evento al `MediaRenderer`. En este caso sólo cuando la ventana de la imagen se cierre.

El display con la imagen se crea a partir de la invocación de la acción *Play()*. Cuando se llega al método play() de la clase MediaRenderer, ésta verifica que la URI identifica una imagen, e invoca al método createImageDisplay() que crea la ventana de visualización. Este método también es invocado durante una presentación de imágenes, en la que el *Control Point* (en el caso de Cidero) invoca la acción Play() cada vez que se quiere cambiar la imagen por otra. En este caso, el método no crea una nueva ventana sino que elimina la etiqueta (JLabel) anterior y añade una nuevo donde pinta la nueva imagen.

La ventana se cerrará en respuesta a la acción Stop() o si se pulsa el botón que por defecto provee para ello el frame.

- ImageFrame: clase que proporciona la ventana donde se visualiza la imagen. La imagen se pinta en un componente JLabel que será añadido a esta ventana junto con otra etiqueta en la que se incluye el título de la imagen.

- ImageFrame.ImageLabel: clase interna de ImageFrame. Proporciona el componente JLabel donde se va a pintar la imagen. Su método principal, paintComponent(), es el encargado de pintar la imagen.

3. homeAV.media.rtp: contiene el conjunto de métodos y clases para el manejo de una sesión RTP y la reproducción de los flujos de audio y video de ésta.

   Clases:

- RTPReceiver: se trata de la clase que maneja la recepción de flujos RTP. Fue creada en base a la clase AVReceive3 que provee Sun Microsystems.
  Implementa varias clases escuchadoras de eventos:

  - *ReceiveStreamListener*, para los eventos que genere el flujo RTP asociado a la sesión. En HomeAV interesan únicamente los generados cuando llega un nuevo flujo de un participante de la sesión, para crear un nuevo Player para éste, y cuando finaliza, para cerrarlo y cerrar la sesión.

  - *SessionListener*, para los eventos asociados con el estado de la sesión. En HomeAV interesa únicamente el evento generado cuando se incorpora un nuevo participante a la sesión.

  - *ControllerListener*, para enterarse de los eventos relacionados con el/los Player que reproducen los flujos RTP y actuar en consecuencia.

  La creación de una sesión RTP comienza con la invocación de la acción SetAVTransportURI(), lo que provoca que se invoque el método createPlayback() de MediaRenderer. Éste hace la llamada al método createRTPSessionLabel() de esta clase. En él se crea una etiqueta con los datos necesarios para abrir una sesión RTP, estos son: dirección IP, puerto y ttl. En caso de que HomeAV esté configurado para la apertura de dos sesiones, una para audio y otra para video, se creará una segunda etiqueta con las mimos parámetros anteriores excepto el puerto que consistirá en dos números más que el valor para la primera sesión.

  Posteriormente, en respuesta a la acción Play(), se invoca el método play() de MediaRenderer. Éste método llama al método initializeRTPSession() de esta clase. En éste se crea un manejador RTP por cada sesión a abrir, RTPManager, utilizando una instancia de la clase RTPSocketAdapter para crear la conexión y utilizando los parámetros anteriormente descritos.

```
        /** Initialize the RTP session.
         * This method creates and initialize a 'RTPManager' instance for
         * every 'SessionLabel' instance using a 'RTPSocketAdapter'instance.
        public boolean initializeRTPSession() {
              try {
                    mgrs = new RTPManager[sessions.length];
                    instances = new Vector<Instance>();
                    // Open the RTP sessions
                    for (int i = 0; i < sessions.length; i++) {
                          // Parse the session addresses.
                          logger.info("RTP session: Open RTP session for: addr:
                          " + sessions[i].addr + " port: " + sessions[i].port +
                          " ttl: " + sessions[i].ttl);
                          mgrs[i] = (RTPManager) RTPManager.newInstance();
                          mgrs[i].addSessionListener(this);
                          mgrs[i].addReceiveStreamListener(this);
                          // Initialize the RTPManager with the RTPSocketAdapter
                          mgrs[i].initialize(new
                          RTPSocketAdapter(InetAddress.getByName(sessions[i].add
                          r), sessions[i].port, sessions[i].ttl));
                          BufferControl bc =
                          (BufferControl)mgrs[i].getControl("javax.media.control
                          .BufferControl");
                          if (bc != null)
                                bc.setBufferLength(350);
                    }
              } catch (Exception e){
                    logger.warning("RTP session: Cannot create the RTP Session:
                    " + e.getMessage());
                    return false;
              }
              isRTPReceiverAlive = true;
              prefetchEventSent = false;
              startEventSent = false;
              controllerErrorEventSent = false;
              remotePayloadEventSent = false;
              return true;
        }
```

Cuadro de texto 3.4. Método `initializeRTPSession()` de `RTPReceiver`

Una vez que se configura la conexión la sesión ya esta abierta. Se ha de esperar a que se reciba el evento de llegada de un flujo por medio del método que define la clase escuchadora correspondiente de las comentadas en líneas anteriores y entonces crear el `Player` que lo manejará. Esto ocurrirá por cada uno de los flujos de cada sesión abierta. A partir de aquí, se podrá actuar sobre la reproducción invocando los controles usuales comentados que proporciona cada `Player`, y escuchar sus eventos para actuar en consecuencia.

La conexión, y por tanto la sesión se cerrará, como resultado de la invocación del método `Stop()` o bien cuando se reciba un evento del flujo que así lo indique(`ByeEvent`).

`RTPReceiver` contiene dos clases internas:

- `SessionLabel`: para crear instancias que definen la etiqueta con los parámetros de la sesión y que será utilizada para crear la sesión RTP.

- `Instance`: esta clase es utilizada para manejar cada `Player` de la sesión. Asocia cada `Player` con el flujo que reproduce. Una instancia de esta clase está compuesta por un `Player` y su flujo correspondiente. Esto supone que se haya que crear tantas instancias de ella como flujos a reproducir, en este caso dos (de audio y de video).

- `RTPSocketAdapter`: es la clase que define las conexiones de la sesión RTP. Fue adaptada de la clase del mismo nombre que provee Sun Microsystems. Únicamente se modificó la forma de obtener la IP local donde se abrirán los *sockets*. Implementa a la interfaz *`RTPConnector`* que provee JMF, para crear conexiones basadas en *sockets* UDP. Estos *sockets* serán utilizados por el `RTPManager` para enviar y recibir mensajes de control RTCP, y recibir el flujo RTP.

  Posee dos clases internas:

  - `SockInputStream`: define una fuente de flujo de entrada, el que se recibe. Utilizada por el `RTPManager` tanto para RTP como para RTCP.

  - `SockOutputStream`: define una fuente de flujo de salida, el que se envía. Utilizada en este caso por el `RTPManager` para RTCP.

3.3.4 Diagramas de secuencia

A fin de poder comprender mejor el funcionamiento de la aplicación, se han creado los siguientes diagramas UML de secuencia de los principales procesos de la misma. Se recomienda consultar el “ANEXO III: Diagramas UML” donde se describe la simbología utilizada.

Estos diagramas muestran de forma esquemática la secuencia de invocación de métodos que realiza el sistema para la implementación de una funcionalidad. Se han incluido las clases de Cidero y Cyberlink que se han considerado más importantes para su comprensión.

Los principales procesos, por su importancia en el diseño de la aplicación, son:

1. Arranque de la aplicación.
2. Recepción de mensajes de solicitud HTTP.
3. Obtención del documento descriptor.
4. Proceso de subscripción a un servicio (*Eventing*).
5. Proceso de invocación de una acción:
   a. `SetAVTransportURI()`: reproducción HTTP.
   b. `Play()`: reproducción HTTP.
   c. `SetAVTransportURI()`: *streaming* RTP.
   d. `Play()`: *streaming* RTP.
   e. `SetNextAVTransportURI()`.
6. Proceso de notificación de eventos.

1. Arranque de la Aplicación:

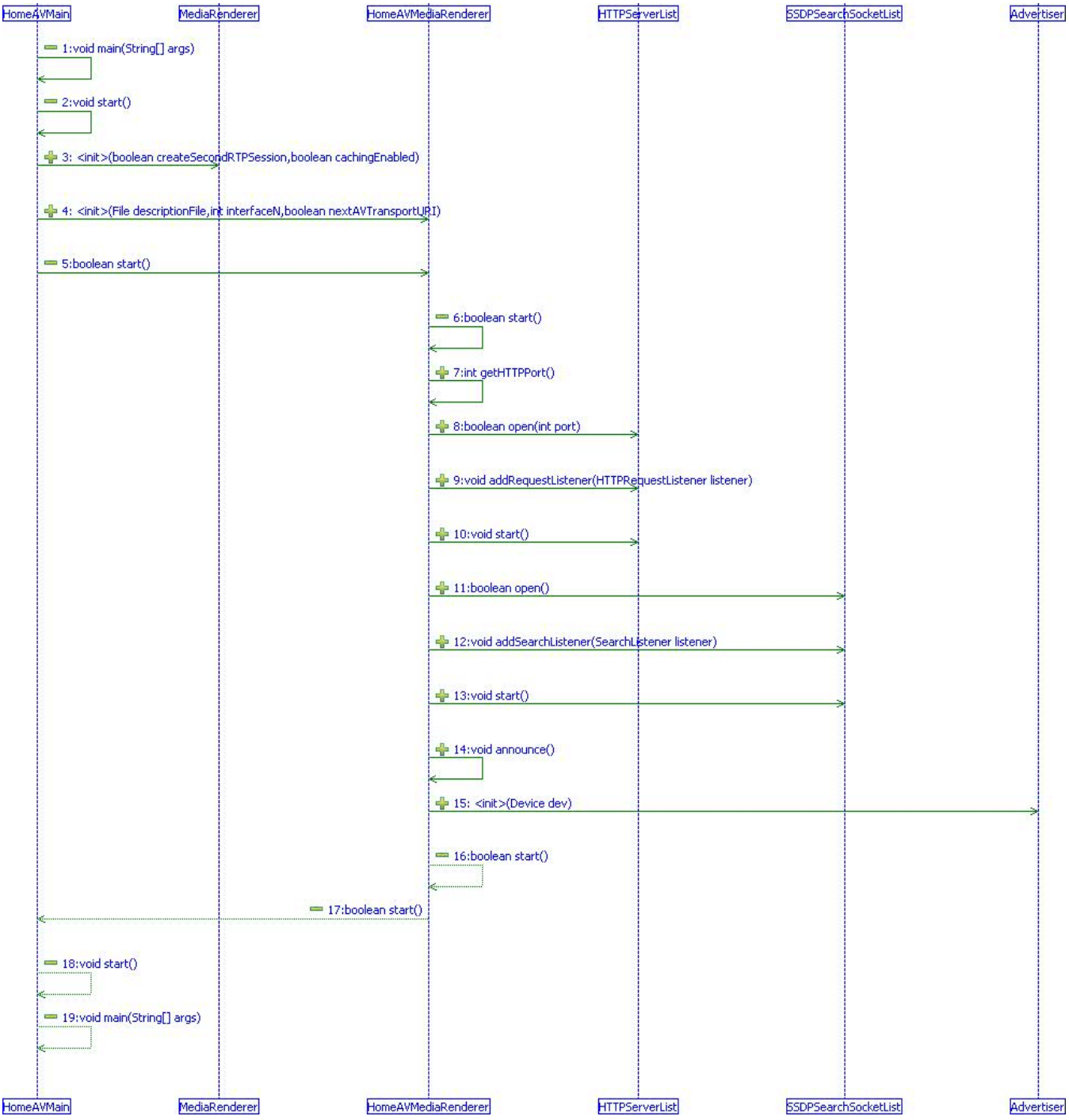


Figura 3.4. Diagrama de secuencia del arranque de la aplicación

La secuencia comienza con la invocación del método `start()` de arranque de la aplicación. En este, como se ya se dijo con anterioridad, se crean las instancias de la clase que implementa el dispositivo UPnP, `HomeAVMediaRenderer`, y la clase que implementa la funcionalidad de la renderización, `MediaRenderer`. Estas clases a su vez crear y configurar las instancias de las respectivas clases que las contiene a fin de estar disponibles para su cuando se requiera.

Continuando la ejecución del método `start()`, la siguiente instrucción es la invocación al método del mismo nombre, `start()`, de `HomeAVMediaRenderer`. Como consecuencia, se desencadena todo el proceso de inicialización y anuncio del dispositivo. Este proceso implica la

creación de la lista de servidores HTTP y *sockets* para la búsqueda SSDP, en este caso con un único elemento ya que HomeAV usa una sola interfaz de red para la comunicación. Además se crea el hilo `Advertiser` encargado del envío de mensajes de anuncio periódicos a la red, tal y como define el estándar UPnP.

Por tanto, cuando un *Control Point* recibe un mensaje de anuncio del dispositivo, ya puede obtener la información de su descripción a partir de la URL que se especifica en el mensaje SSDP recibido. Si un dispositivo no es descubierto tras este mensaje de anuncio, normalmente porque el *Control Point* haya arrancado posteriormente, recibirá un mensaje SSDPSearch de éste y procederá a enviarle una respuesta SSDP con la URL de su descriptor.

El servidor HTTP, por medio de un *socket* HTTP, será el que reciba todos los mensajes de solicitud que envíe el *Control Point* utilizando este protocolo. Entre estos mensajes están la solicitud del descriptor del dispositivo, las solicitudes de subscripción y las invocaciones de las acciones (Más información en “ANEXO I: Cidero/Cyberlink”).

2. Recepción de mensajes de solicitud HTTP:

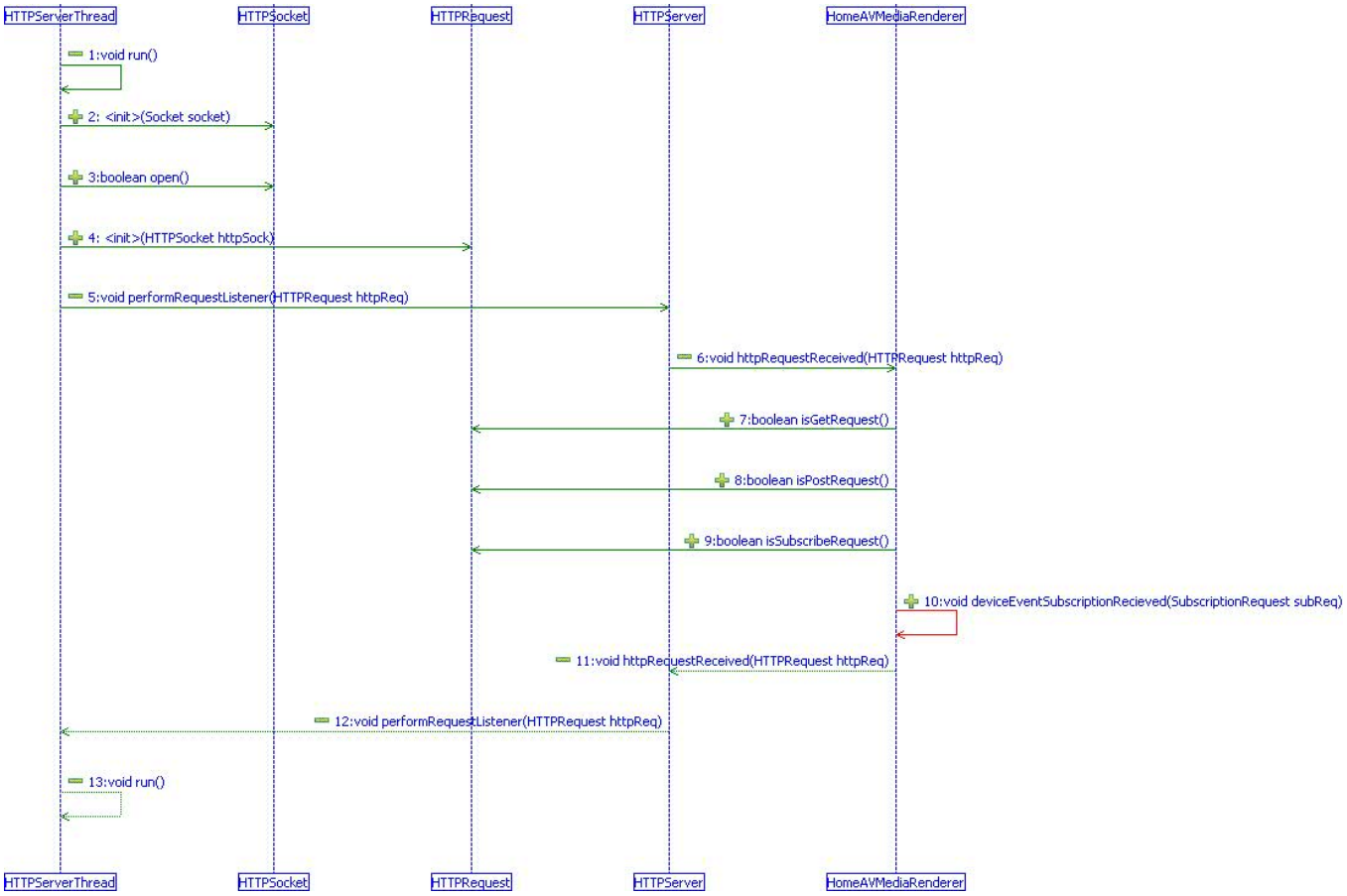


Figura 3.5. Diagrama de secuencia de recepción de solicitud HTTP

Como se ha dicho, la recepción de solicitudes es gestionada por el mismo hilo de ejecución del servidor HTTP. Esto supone que el proceso sea análogo para cada uno de los tres casos mencionados anteriormente.

Se recibe una solicitud en el *socket* HTTP. El servidor informa de ello a los escuchadores de solicitudes http (*`HTTPRequestListener`*), en este caso la clase *Device*, que por herencia figura en el diagrama como su clase derivada (`HomeAVMediaRenderer`). Esto provoca la invocación del método `httpRequestReceived()`. En este método se comprueba de que tipo de solicitud se trata: para la obtención de un descriptor o para la invocación de una acción o un mensaje de alta/cancelación/renovación de la subscripción al servicio (*Eventing*).

3. Obtención del documento descriptor:

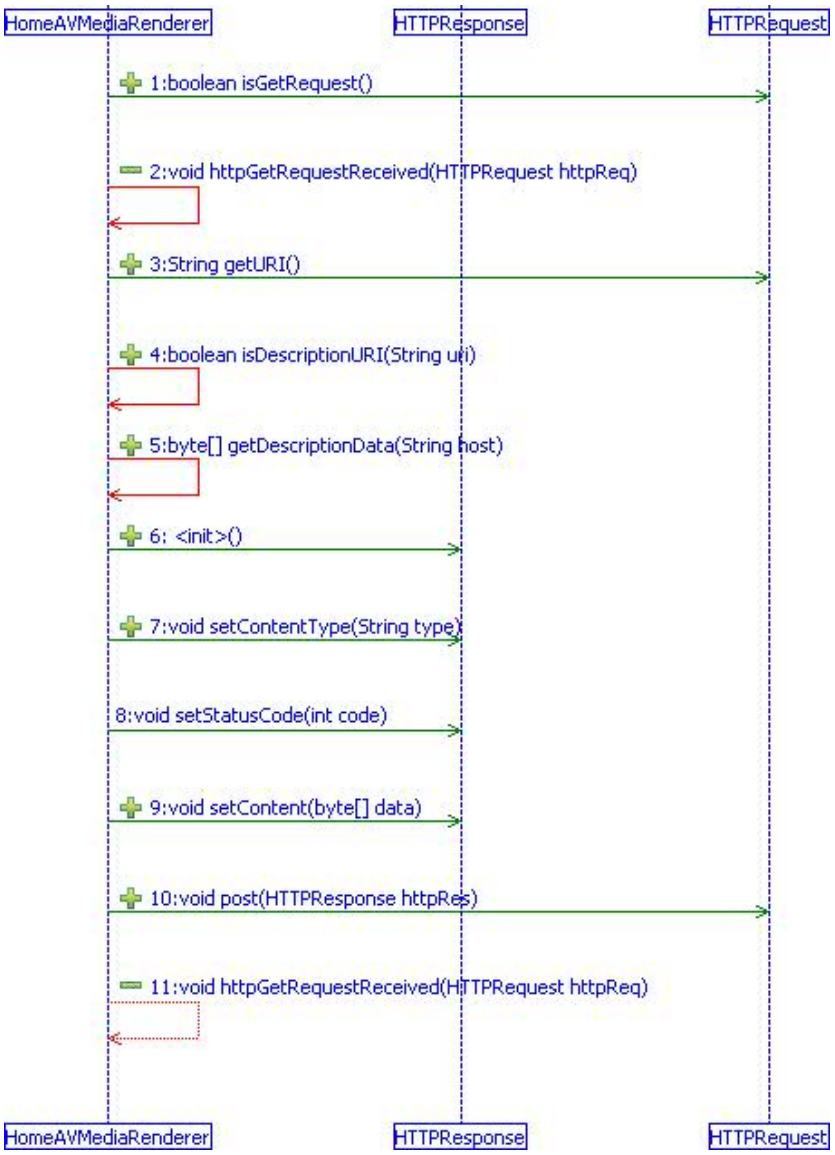


Figura 3.6. Diagrama de secuencia de la obtención del descriptor

Una vez que el dispositivo ya ha sido descubierto, el *Control Point* puede obtener el descriptor del mismo a través de la URL facilitada en el mensaje SSDP de descubrimiento.

Tras ser recibido el mensaje de solicitud y clasificado por su tipo, se invoca al método `httpGetRequestReceived()` de `Device` (por medio de su clase derivada de HomeAV, `HomeAVMediaRenderer`). Este método se encarga de obtener el contenido del descriptor para ser enviado en un mensaje `HTTPResponse` de respuesta.

4. Subscripción a un servicio:

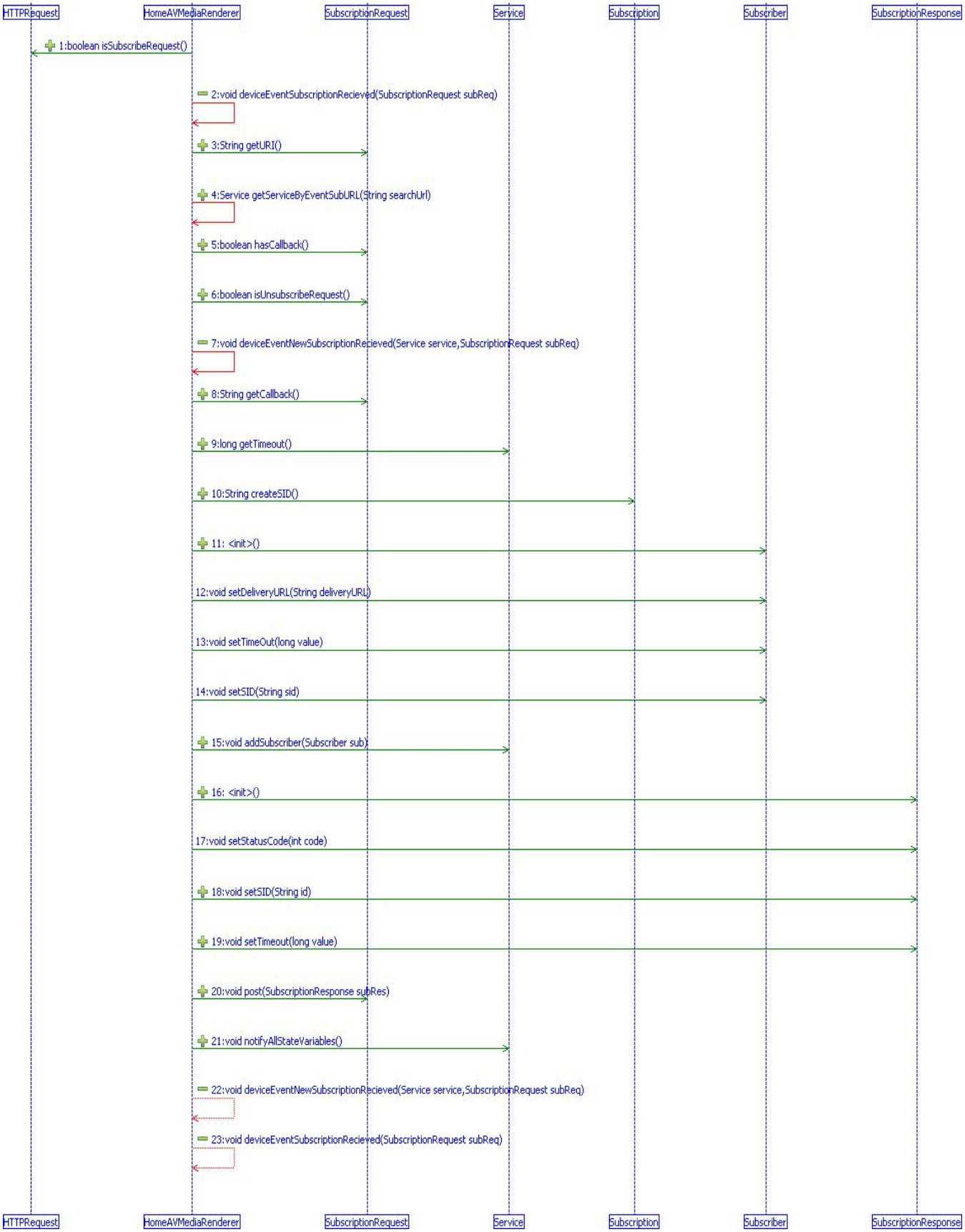


Figura 3.7. Diagrama de secuencia de la subscripción a un servicio

Tras la obtención por parte del *Control Point* del descriptor del dispositivo y, a partir de las URLs que figuran en este, los de los servicios, ya dispones de la información necesaria para solicitar la subscripción a un servicio para la recepción de los eventos generados en este por cambios en su modelo de estado.

Una vez que se comprueba que se trata de una solicitud relacionada con la subscripción a un servicio, se invoca al correspondiente método, `deviceEventSubscriptionReceived()` de `Device` (`HomeAVMediaRenderer`). En éste se comprueba el tipo de solicitud de subscripción de que se trata, esto es: para una nueva subscripción, para cancelarla o para renovarla. El caso del diagrama es para la primera opción.

Al ser una nueva subscripción, se invoca al correspondiente método de `Device` también, `deviceEventNewSubscriptionReceived()`. El algoritmo de este método procesa la solicitud configurando los parámetros necesarios conforme a la especificación del protocolo GENA y que serán incluidos en el mensaje de respuesta. Tras esto, este mensaje se envía al *Control Point* e inmediatamente después se le notifican las variables de estado del servicio con sus valores actuales.

5. Proceso de invocación de una acción: Reproducción HTTP:

Figura 3.8. Diagrama de secuencia de invocación a una acción

En el diagrama anterior se muestra el proceso de invocación de una acción, en este caso `SetAVTransportURI()`. Desde que se comprueba que el mensaje de solicitud HTTP recibido es del tipo SOAP, se ejecuta una serie de invocaciones de métodos anidados de la clase `Device`, que van clasificando el tipo de solicitud SOAP recibida (*Query* o *Action*) y comprobando que ésta es correcta. Cuando la ejecución alcanza el método `deviceActionControlReceived()`, ya se ha verificado que se trata de una acción y se crea una instancia de *Action* a partir del nombre de la acción solicitada y sus argumentos.

Esta instancia es usada para avisar a los escuchadores de acciones (*`ActionListener`*) de la solicitud. Esto provoca la invocación del método `actionControlReceived()` de la clase escuchadora, cualquiera que extienda de `AbstractService`,en este caso `AVTransportService`. Este método posee un algoritmo, que en este caso realiza una llamada al método del servicio que la extiende que tiene el mismo nombre que la acción.

Se procederá de forma análoga para cualquier otra acción del resto de servicios implementados, *RenderingControl Service* y *ConnectionManager Service*.

- `SetAVTrasnportURI()`: reproducción HTTP.

Figura 3.9. Diagrama de secuencia de la acción `SetAVTransportURI()` (HTTP)

En este diagrama se muestra la implementación en HomeAV de la acción `SetAVTransportURI()` para el caso en el que se va a reproducir audio o video obtenido en remoto mediante HTTP. En primer lugar se obtienen los argumentos de entrada de la acción, en este caso la URI que localiza al contenido y un cadena de texto con información de meta-datos del mismo.

A partir de los meta-datos se obtiene la duración y el título fichero, que serán almacenan en sendas variables de la clase `MediaRenderer` y que posteriormente serán recuperadas para ser añadidos en la ventana de la aplicación.

Con la URI como argumento de entrada se invoca al método `createPlayback()` de `MediaRenderer` para que se proceda con la creación de la renderización. El algoritmo de éste método identifica el tipo de contenido asociado que se va a tratar a partir de la URI, es decir, si es audio/video obtenido mediante HTTP, o es una imagen, o es una URI para la creación de una sesión RTP. En función de esto invoca el método de las correspondientes subclases manejadoras para que continúen el proceso específico requerido. Este diagrama trata sobre el primer caso, por lo que se invoca al método `createPlayer()` de la subclase `HTTPRenderer`, que simplemente crea el `Player` asociado al contenido que se desea reproducir. Cuando acaba este proceso, se continua la ejecución del método `actionSetAVTransportURI()` que actualiza las variables de estado asociadas la URI y a los meta-datos.

- `Play()`: reproducción HTTP.

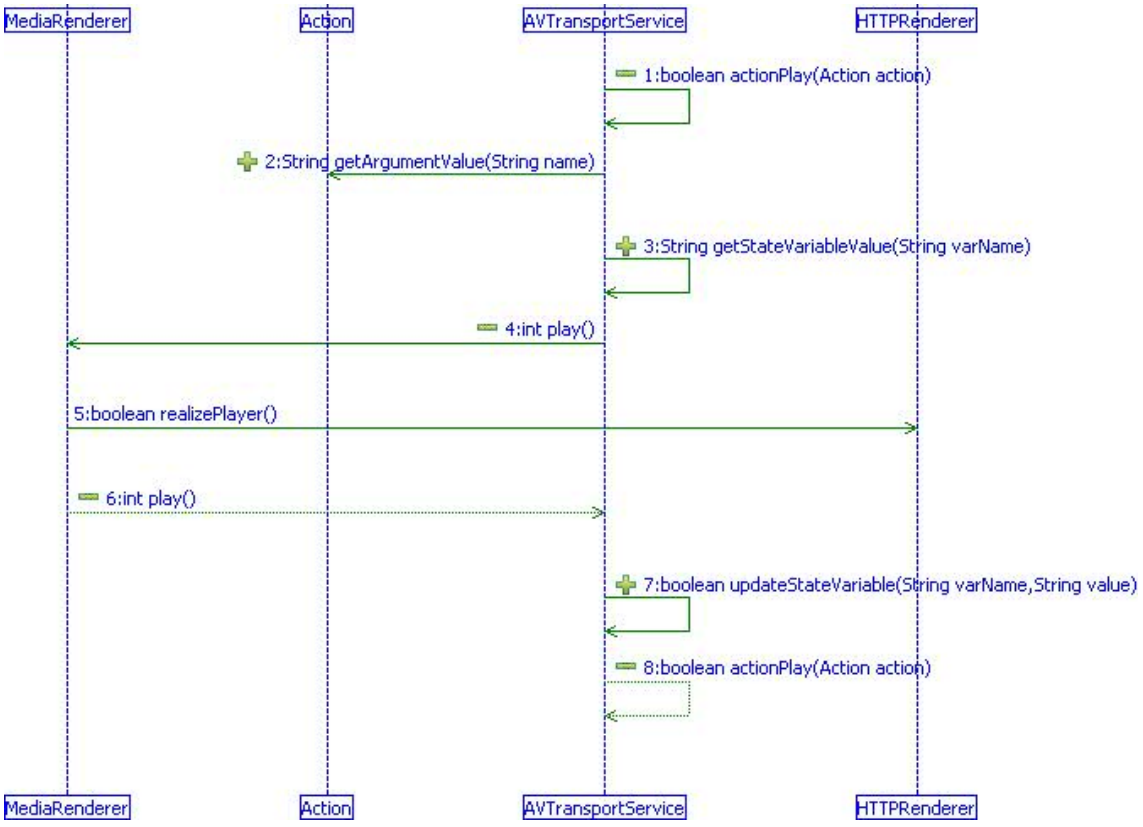


Figura 3.10. Diagrama de secuencia de la acción `Play()` (HTTP)

Como ya se ha explicado, seguidamente a la invocación de `SetAVTrasnportURI()`, el *Control Point* invoca la acción `Play()`. En su implementación se llama al método del mismo nombre de la clase manejadora `MediaRenderer`. Como en el apartado anterior, aquí también se procederá de forma distinta en función del tipo de contenido asociado, llamando al método que corresponde de su subclase manejadora. En este caso se invoca al método `realizePlayer()` de `HTTPRenderer`. Este método inicia al `Player` creado anteriormente, permitiendo que vaya pasando por los distintos estados hasta comenzar la reproducción. Cuando vuelve la ejecución al método de la acción, `actionPlay()`, se actualiza la variable de estado `TransportState` al valor *TRANSITIONING*,

indicando un estado intermedio hasta que comience la reproducción. Cambiará a estado *PLAYING* cuando se reciba el evento del `Player` que indique que ha alcanzado el estado *Started* y ya comienza la reproducción.

- *SetAVTransportURI()*: *streaming* RTP.

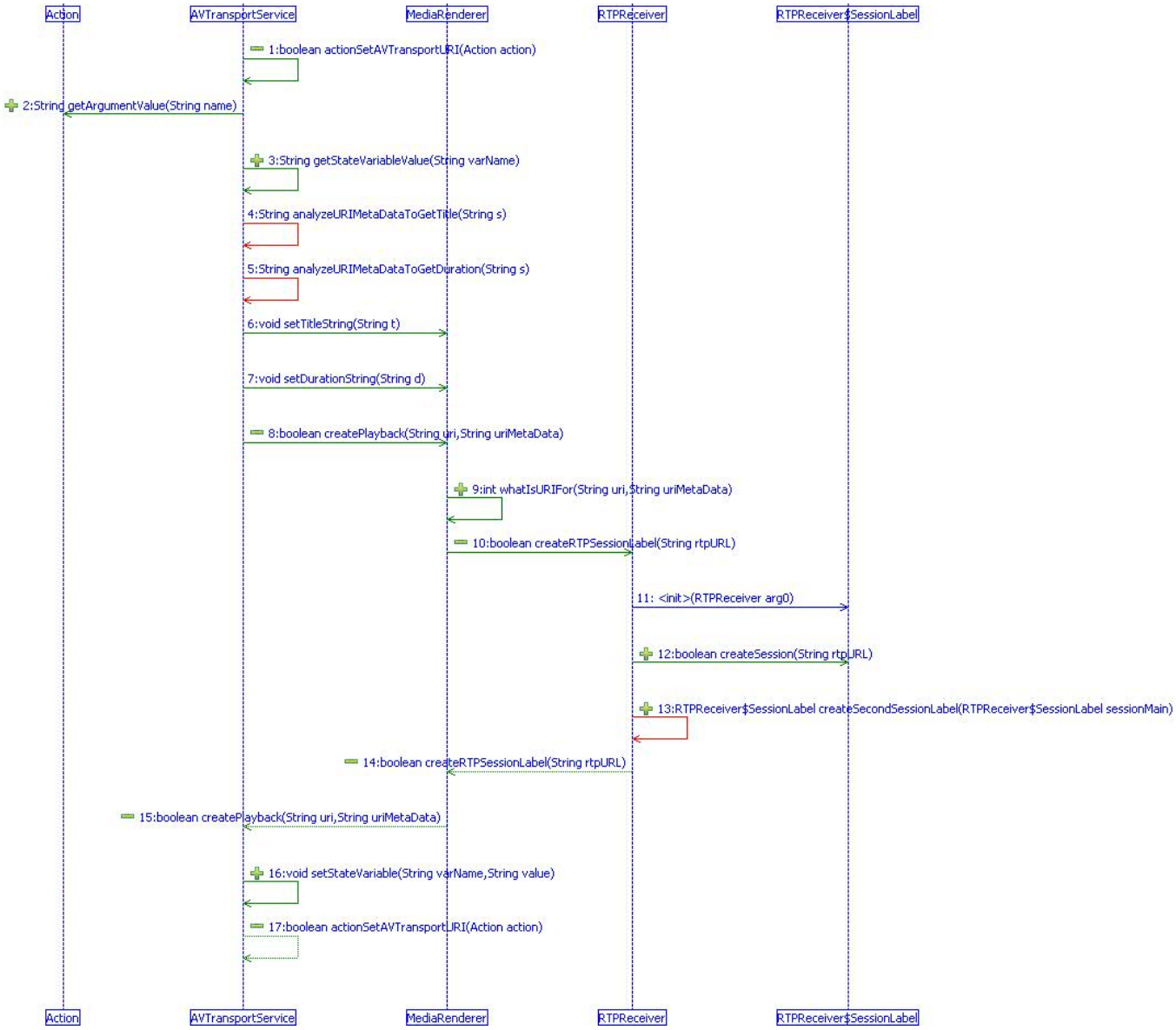


Figura 3.11. Diagrama de secuencia de la acción `SetAVTransportURI()` (RTP)

De forma análoga al caso HTTP, se procede con la implementación de la acción `SetAVTransportURI()` para *streaming* RTP. A diferencia del anterior, en este caso se invoca al método `createSession()` de la correspondiente subclase manejadora, `RTPReceiver`. A partir de la URI obtenida como argumento de la acción, se crea la sesión o sesiones RTP en función de si se va a recibir audio y video, y si está configurado para ello el parámetro correspondiente del archivo de configuración. En este proceso simplemente se crean las etiquetas de la/s sesiones con lo datos de cada una (IP, puerto y ttl).

- `Play()`: *streaming* RTP.

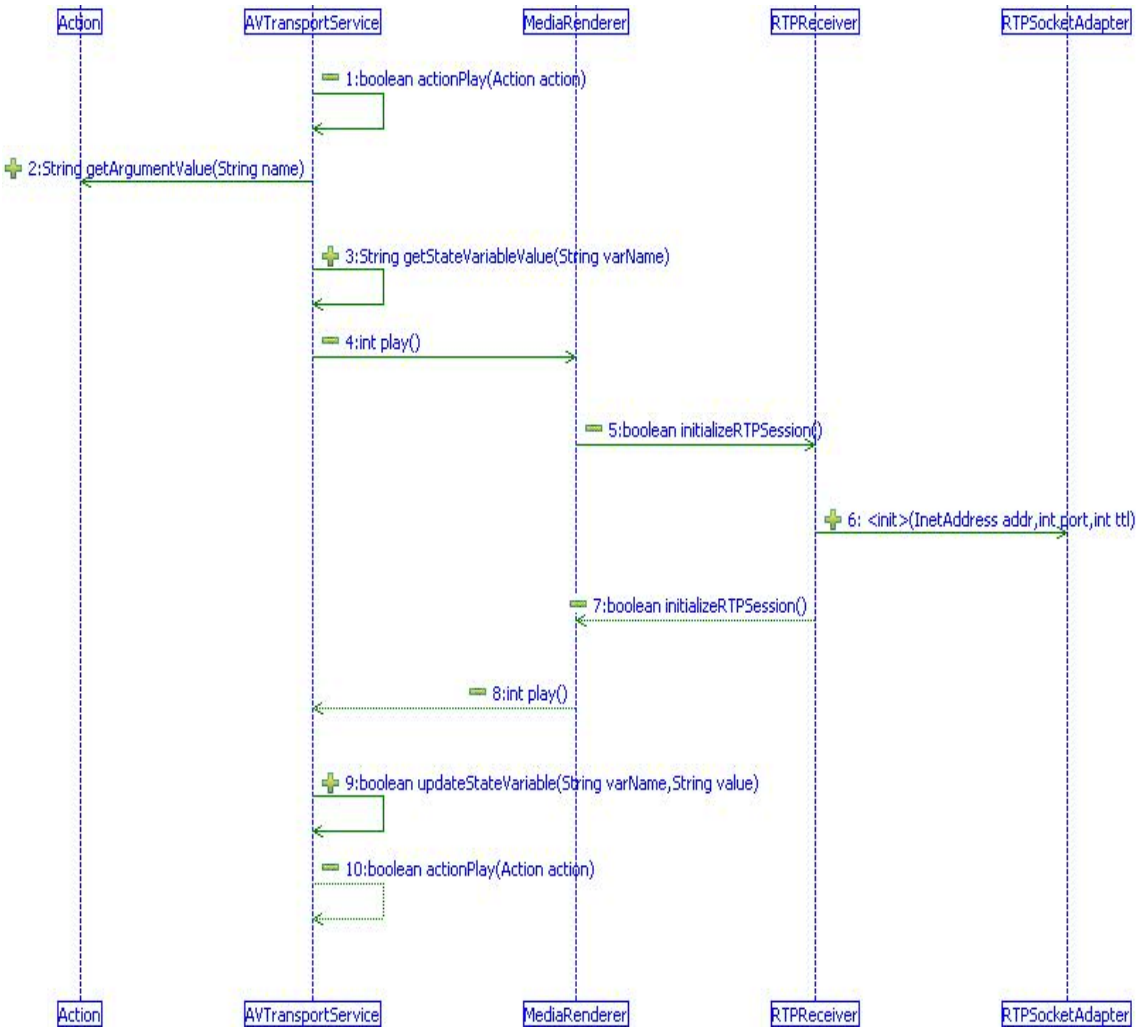


Figura 3.12. Diagrama de secuencia de la acción `Play()` (RTP)

En la invocación de la acción `Play()` también se procede de forma análoga al caso HTTP. En el método `play()` de `MediaRenderer` se invoca al método `initializeRTPSession()` de la correspondiente subclase manejadora, `RTPReceiver`. El algoritmo de este método crea la/s sesiones RTP a partir de la información de las etiquetas creadas en el anterior proceso, creando un manejador de RTP para cada una.

Un manejador RTP es creado utilizando un objeto `RTPSocketAdapter`, con el que crear y abrir las conexiones. Hecho esto, la ejecución vuelve al método `actionPlay()` donde, como en el caso anterior, se actualiza la variable de estado `TransportState` al estado *TRANSITIONING* hasta que, tras recibir el/los flujos RTP con el contenido, se creen los objetos `Player` asociados y envíen el correspondiente evento indicando que han alcanzado el estado *Started*. En este momento se cambiará el estado de la renderización a *PLAYING*.

- `SetNextAVTransportURI()`:

Figura 3.13. Diagrama de secuencia de la acción `SetNextAVTransportURI()`

Como se ha dicho cuando se expusieron las características de HomeAV en este capítulo, la aplicación está configurada para que un `Player` pueda realizar *caching*, es decir, el almacenamiento local del contenido previo a su reproducción. Esto está disponible y resulte útil, cuando se va a renderizar audio y video obtenido mediante el protocolo de transporte HTTP (ver apartado sobre streaming y protocolo RTP en el capítulo “Marco teórico”).

La acción `SetNextAVTransportURI()` esta realmente indicada para este proceso, ya que permite que mientras un `Player` esté en ejecución reproduciendo un medio, se cree otro que inicie el proceso de descarga. Para el mayor número de casos de uso de la aplicación (reproducción de audio MP3, secuencias de video,…), se consigue que una vez finalizado la primera reproducción esté ya disponible el segundo `Player` para iniciar la siguiente de inmediato.

El método `perpareNextPlayback()` de `MediaRenderer` recibe la URI del siguiente recurso en su invocación en el método que implementa la acción `actionSetNextAVTransportURI()`. A partir de su URI determina si el procedimiento es aplicable al contenido. Una vez confirmado que se trata de audio o video la que se accede mediante HTTP, se procede a llamar al método `createPlayer()` de la correspondiente subclase manejadora `HTTPRenderer`. Se trata del mismo método invocado en la implementación de la acción `SetAVTransporURI()`, pero en este caso el algoritmo no sólo crea el `Player`, sino que lo inicia hasta que alcance el estado de *Realized* en el que ya se ha completado la descarga del contenido.

6. Proceso de notificación de eventos:

Figura 3.14. Diagrama de secuencia de la notificación de eventos

El diagrama anterior corresponde a un ejemplo de uno de los procesos más importantes de la aplicación. Se trata de la hacer llegar a la implementación del servicio *AVTransport Service* una serie de eventos, relacionados con la renderización, que le sirvan para modelar su estado, principalmente actualizando alguna variable de estado. En este caso se informa al servicio de que el `Player` para audio/video HTTP ha alcanzado el estado *Started* y comienza la reproducción. En consecuencia, se actualizará la variable de estado `TransportState` al valor *PLAYING*.

Primeramente, los eventos son escuchados en las respectivas subclases manejadoras `HTTPRenderer`, `ImageRenderer`, o `RTPReceiver` que implementan distintas interfaces escuchadoras de eventos como se dijo cuando se habló de la estructura de clases de HomeAV. Posteriormente se invoca al método de la clase `MediaRenderer` definido para los eventos de la subclase en cuestión, en este caso `httpRendererEventReceived()`, cuyo algoritmo permite la actualización de sus propias variables conforme a la nueva situación, y llama el método `notifyRendererEvent()`, que notifica el evento a todos los escuchadores de eventos de la clase por medio de la invocación del correspondiente método del escuchador `mediaRendererEventReceived()`. En realidad sólo hay un escuchador, `AVTransportService`, que utiliza este método para actualizar las variables de estado del servicio, principalmente la variable `TransportState`.

Así pues, los eventos que recibe un escuchador de eventos de `MediaRenderer` son:

- `StartEvent` → proviene de un `Player` creado en la clase `RTPReceiver` o en la clase `HTTPRenderer`, cuando éste alcanza el estado *Started* y comienza la reproducción.

- `EndOfMediaEvent` → proviene del `Player` creado en la clase `HTTPRenderer`, cuando alcanza el final del medio.

- `ByeEvent` → proviene de la clase `RTPReceiver`, cuando llega el evento del mismo nombre generado por el flujo recibido cuando este finaliza.

- `ImageCloseEvent` → proviene de la clase `ImageRenderer`, y se genera cuando el usuario cierra la ventana de visualización de la imagen

- Eventos relacionados con errores producidos en la ejecución de un `Player`:

    - Errores correspondientes a un `Player` creado en la clase `HTTPRenderer`: `ControllerErrorEventFromPlayer` (problema en el `Player` usado en la reproducción actual), `ControllerErrorEventFromPlayerNextURI` (problema en el `Player` creado para la siguiente reproducción), `SeekFailedEvent` (problema al establecer un punto concreto temporal en el medio para reproducir desde ahí)

    - Errores correspondientes a un `Player` creado en la clase `RTPReceiver`: `ControllerErrorEventFromRTPPlayer` (problema en alguno de los dos `Player` o el `Player` que está siendo usado para la reproducción de un flujo RTP).

3.3.5 Estados de la renderización

Una de las tareas más importantes del desarrollo de HomeAV ha sido la definición de las transiciones a los distintos estados de la renderización. Esto se hace imprescindible ya que el *Control Point* debe conocer en todo momento el estado en que se encuentra el dispositivo.

Como se dijo en el capítulo “Marco teórico”, el servicio *AVTransport Serivce* provee una variable de estado, `TransportState`, para definir el estado en que se encuentre el transporte del contenido multimedia. En HomeAV estos estados se interpretan como los estados de la renderización.

Las modificaciones en esta variable vienen determinadas por la invocación y ejecución de alguna de las acciones del servicio que controlan la renderización (`SetAVTransportURI()`, `Play()`, `Stop()`,…).

En el apartado anterior se comentaron alguna de estas modificaciones. A continuación se estudiará con más detalle los distintos estados, y por tanto, valores de la variable en los que se puede encontrar la renderización en HomeAV.

Considerando los tres tipos de contenidos que se pueden manejar (audio/video obtenido mediante HTTP, imágenes, o flujo RTP), la invocación de un método modificará de una u otra forma el valor de la variable de estado.

Estos valores son:

- *STOPPED* → Parado (Estado por defecto).

- *TRANSITINING* → En proceso de transición entre estados. Este estado sólo tiene sentido cuando se está utilizando un `Player`, ya que éste ha de pasar por varios estados antes de empezar la reproducción (ver apartado sobre JMF del capítulo “Marco Teórico”).

- *PLAYING* → Renderizando. Esto es, si existe un `Player` y está en estado *Started* (reproduciendo). Si se trata de la visualización de una imagen, se considera que se encuentra en este estado cuando se está mostrando la imagen.

- *PAUSED_PLAYBACK* → La renderización esta pausada como consecuencia de la invocación de la acción `Pause()`.

Acorde a la especificación UPnP para el servicio *AVTrasnport Service* y asumiendo ciertas limitaciones que se mencionan en el capítulo "Conclusiones", se definió el siguiente diagrama de estados:

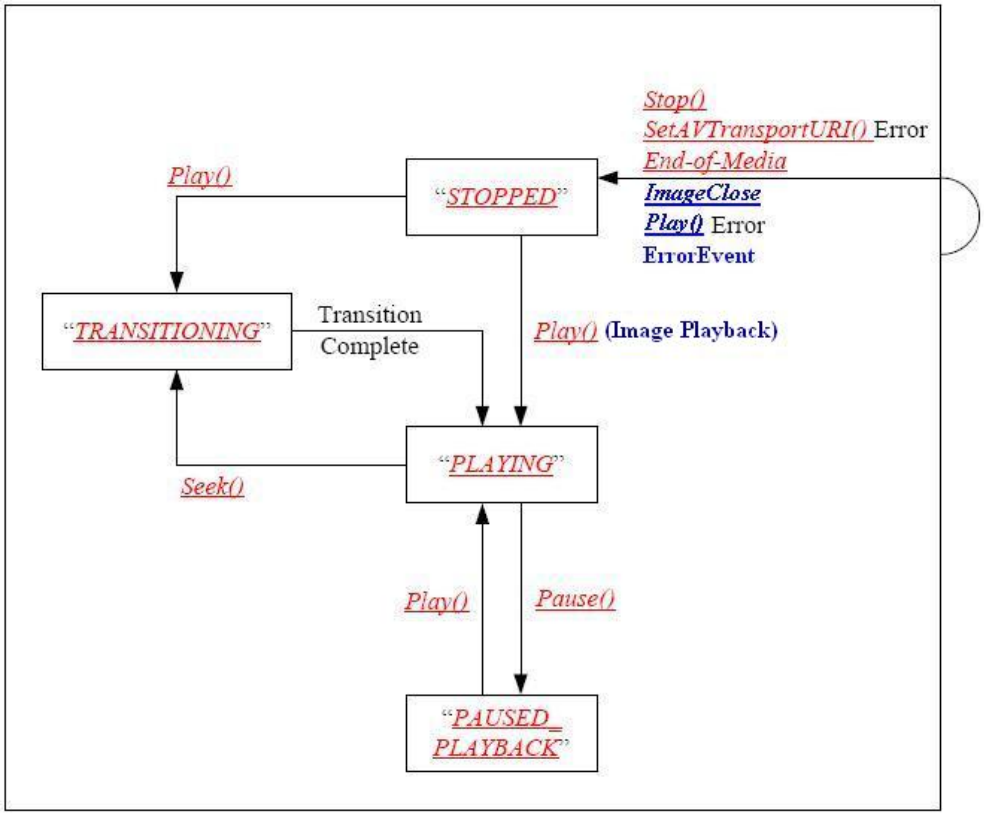


Figura 3.15. Diagrama de estados de la renderización

Los caracteres escritos en color azul son variaciones introducidas en las causas que generan las transiciones entre estados y son acordes al comportamiento de la aplicación.

El estado por defecto en HomeAV es *STOPPED*. La variable `TransportState` toma este valor desde el inicio y siempre que ocurra alguno de los siguientes casos:

- Si se recibe un evento del tipo `EndOfMediaEvent` del `Player` (caso HTTP), indicando que se ha alcanzado el final del fichero de audio/video, y siempre que no esté programada una próxima reproducción.

- Si se recibe un evento del tipo `ByeEvent` del flujo RTP indicando el fin de la sesión.

- Si se recibe un evento de cierre de la imagen (`ImageClosedEvent`).

- Siempre que ocurra un error. Ya sea en la invocación de las acciones `SetAVTransportURI()` o `Play()`, o como resultado de la recepción de un evento provocado por un fallo en el `Player` que esté en ejecución.

- Si se ha invocado la acción `Stop()` para parar la reproducción.

Se pasa al estado *TRANSITIONING* en tres supuestos:

- Se ha invocado la acción `Play()` para el comienzo de una nueva reproducción de contenido obtenido mediante HTTP. Esto es como consecuencia del tiempo que tarda el `Player` desde su creación hasta que alcanza el estado *Started*.

- Se ha invocado la acción `Play()` para la creación de una sesión RTP, como consecuencia del tiempo que transcurre desde la creación de la sesión hasta que es recibido el/los streams, y se crean los correspondientes `Player`.

- Se ha invocado la acción `Seek()` mientras se está reproduciendo el medio. Sólo disponible para reproducciones de audio y video con HTTP como protocolo de transporte. De esta forma se mantiene el estado en este valor hasta que se alcanza el punto deseado en el archivo y se arranca de nuevo la reproducción.

Se alcanza el estado *PLAYING* cuando ocurre uno de los siguientes casos:

- Se recibe un evento del `Player` advirtiendo que ya ha alcanzado el estado *Started* y que por tanto se puede pasar del estado *TRANSITIONING* al *PLAYING*. Esto es válido tanto para el `Player` de la reproducción basada en HTTP, como los posibles `Player` asociados a los respectivos flujos RTP.

- Se invoca la acción `Play()` para la visualización de una imagen. En este caso no se pasa por el estado *TRANSITIONING* ya que no se ha de hacer ningún proceso previo que consuma tiempo.

Se pasa al estado *PAUSE_PLAYBACK* como consecuencia de la invocación de la acción `Pause()`. Y consecuentemente, vuelve la estado *PLAYING* si se invoca la acción `Play()`.

4. Pruebas

En este capítulo se recogen las pruebas realizadas a la aplicación con el objeto de valorar la respuesta del sistema en un escenario real de ejecución y presentar datos de su eficiencia.

4.1 Introducción

Los objetivos que se persiguen con estas pruebas son:

- Mostrar y valorar el tiempo de respuesta de la aplicación (HomeAV) comprendido entre su arranque y su descubrimiento en el *Control Point*.

- Mostrar y valorar el tiempo de repuesta de la aplicación (HomeAV) comprendido entre la solicitud de reproducción de un recurso multimedia y la confirmación del éxito de la misma.

4.1.1 Escenario de pruebas

En la siguiente figura se muestra el escenario sobre el que se realizaron las pruebas de la aplicación.

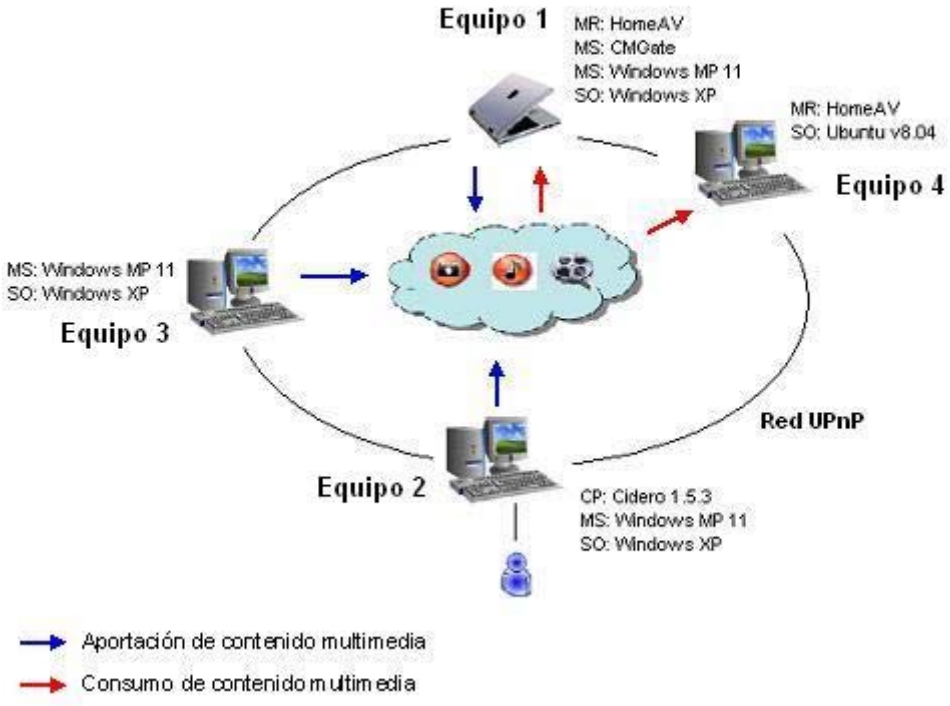


Figura 4.1. Escenario de pruebas

Donde,

MR: *UPnP MediaRenderer* (consumidor de contenidos multimedia)
MS: *UPnP MediaServer* (servidor de contenido multimedia)
CP: *UPnP AV Control Point* (controlador de los dispositivos AV UPnP)

Características del escenario:

El escenario está compuesto por cuatro equipos (PCs) conectados a un router por medio de cables Ethernet de 100Mbps. Cada uno de ellos, está configurado con una IP única en la red UPnP creada. Las características de los equipos son:

	RAM	Procesador	Sistema Operativo
Equipo 1	512MB	1.60GHz	Windows XP HE
Equipo 2	256MB	851MHz	Windows XP Prof
Equipo 3	630MB	800MHz	Windows XP Prof
Equipo 4	256MB	800MHz	Ubuntu 8.04

Tabla 4.1. Características de los equipos utilizados

Un equipo puede contener uno o más dispositivos UPnP, tal y como se muestra en la figura.

Las aplicaciones UPnP utilizadas que integran estos dispositivos son las que se presentaron en el apartado de “Herramientas del Sistema” del anterior capítulo.

La red UPnP resultante permite al *Control Point* (Cidero) listar el contenido de cada uno de los *MediaServer* (Windows Media Player, CyberMediaGate) presentes y seleccionar el *MediaRenderer* (HomeAV) donde reproducir los ítems.

4.1.2 Condiciones de la toma de datos

A continuación se enumeran las condiciones que se establecieron en la toma de datos de las pruebas:

- Se considerarán nulos los retardos de transmisión, ya que el propósito de estas pruebas no incluye la valoración ni del tipo ni del estado de la red.
- El HomeAV utilizado para la toma de datos es el que se dispone en el equipo Windows (Equipo 1).
- La implementación OSGi utilizada es Equinox de Eclipse 3.3.2.
- Aunque se realizaron varias pruebas con IPv6, en las que aquí se analizan se utilizó un escenario IPv4.
- Los resultados obtenidos de dichas pruebas tendrán un carácter orientativo, ya que en todos los procesos intervienen múltiples factores asociados a las capacidades de computación de los equipos que intervienen.

4.2 Caso 1: Fase de Descubrimiento

Se trata de una simple prueba con la que se pretende mostrar la sencillez y rapidez con la que el dispositivo HomeAV se agrega a la red UPnP.

El dato evaluado es el tiempo transcurrido entre la invocación del comando `start` del *Framework* OSGi y la recepción de la solicitud por parte del *Control Point* del documento descriptor de HomeAV.

Procedimiento:

Lo primero es instalar el *bundle* “HomeAV-*bundle*-1.0.0_win.jar” en la implementación de la plataforma que se utiliza (Equinox).

Una vez instalado y obtenido el identificador que le asigna el *Framework* se puede proceder al arranque del *bundle* por medio del comando `start`, lo que provocará el inicio del proceso de descubrimiento del dispositivo y la visualización de su interfaz gráfica.

Al final del proceso de descubrimiento, Cidero visualizará un icono con el *FriendlyName* de HomeAV en el área de su interfaz gráfica en la que se muestran los *MediaRenderer*s.

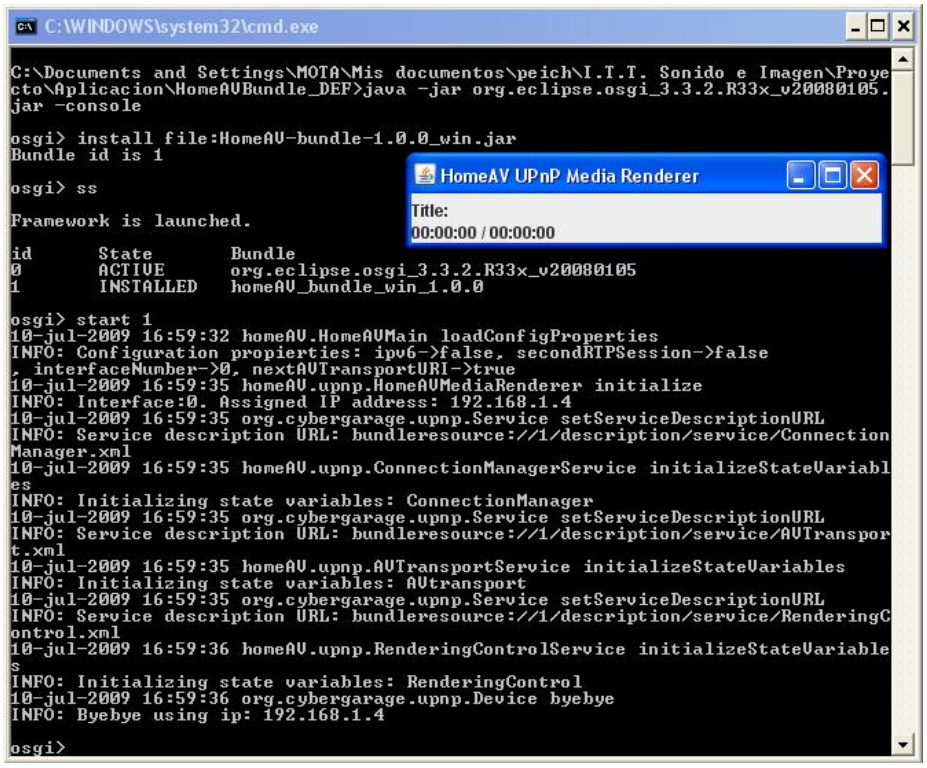


Figura 4.2. Captura de la consola y la interfaz gráfica de HomeAV en la instalación y arranque del *bundle* (Equipo 1)

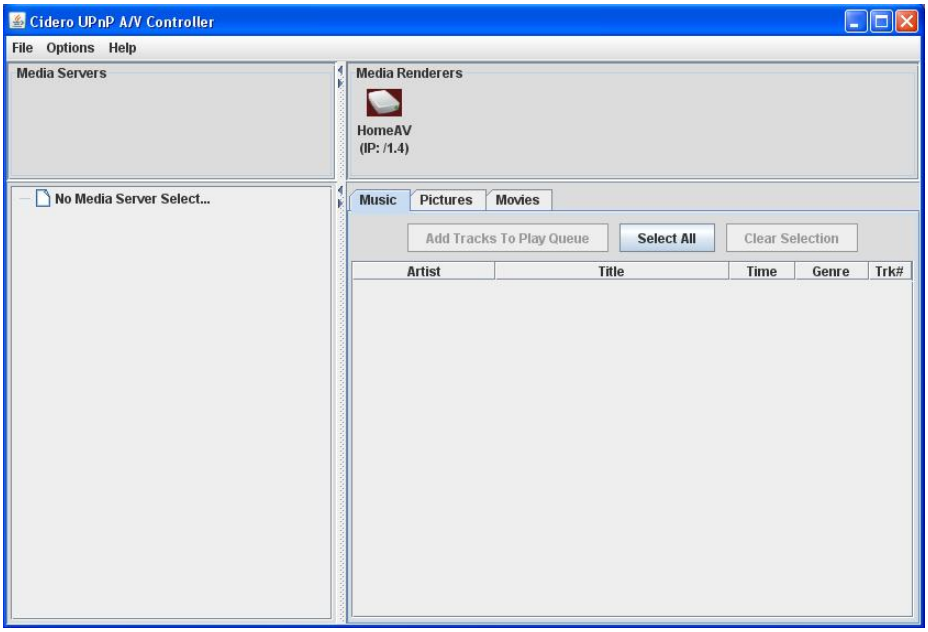


Figura 4.3. Captura de la Interfaz Gráfica de Cidero (Descubrimiento de HomeAV)

Resultados:

El valor promedio obtenido es: **1,610 segundos**

Es decir, con tan solo invocar un comando (`start`), y en poco más de un segundo y medio, el dispositivo HomeAV se configura y pasa a formar parte de la red UPnP de forma automática y transparente para el usuario. A partir de este momento, la aplicación puede ser utilizada por éste a través de la interfaz gráfica del *Control Point*.

4.3 Caso 2: Reproducción de ficheros de audio y video

En esta prueba se analizarán los valores obtenidos del tiempo que transcurre entre la pulsación del botón "*Play*" del panel que provee Cidero para el control de HomeAV, y la confirmación de que la reproducción ha comenzado, esto es, la recepción en *Cidero* del evento generado por el cambio de valor de la ya estudiada variable `TransportState` a "PLAYING".

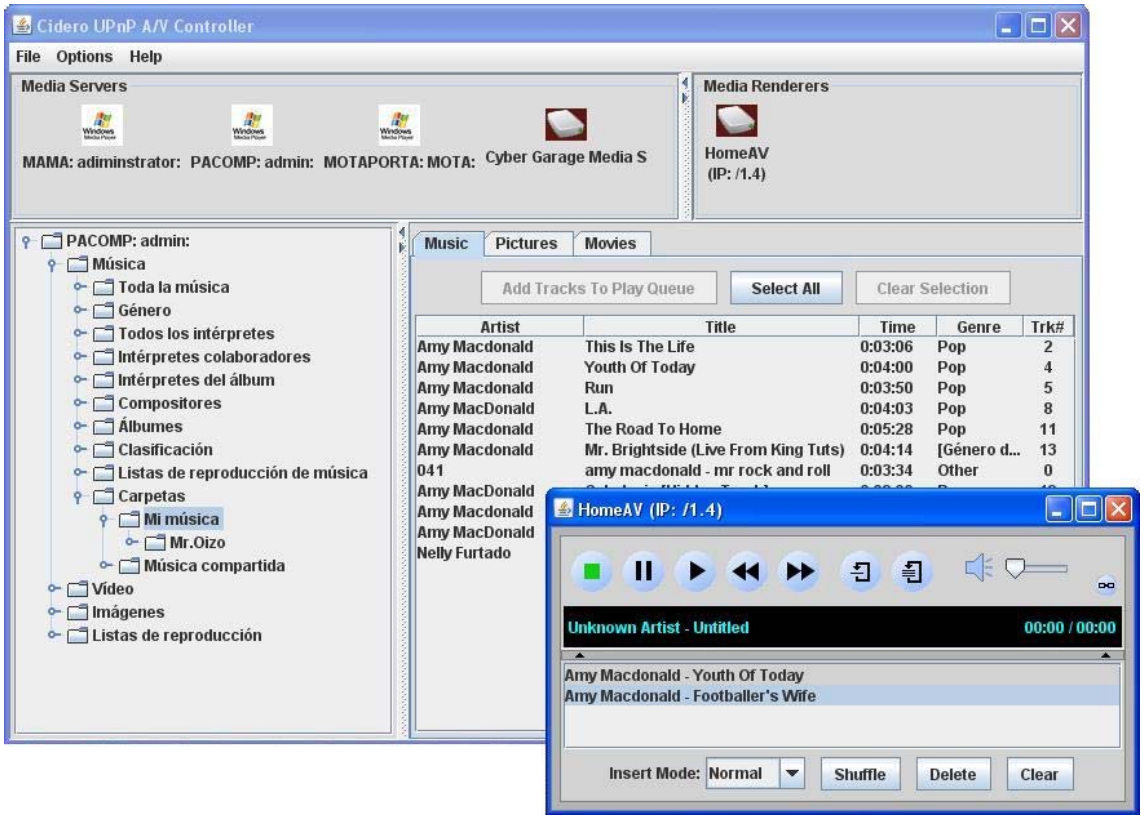


Figura 4.4. Captura de la interfaz gráfica de Cidero y el panel que provee para el *UPnP MediaRenderer*

1. Contenido obtenido por medio del protocolo HTTP:

Para esta prueba se utilizaron los siguientes tipos de fichero de audio y video:

	Tipo	Extensión	Tamaño (kB)	Duración (seg)	Tasa de bits media (kbps)
1	Audio: MPEG-1 Video: MPEG-1	.mpg	5783	86	538
2	Audio: MPEG-1 Layer 3 (MP3) Video: DIVX	.avi	8212	79	832
3	Audio: MPEG-1 Layer 3 (MP3) Video: DIVX	.avi	42938	1246	276
4	Audio: MPEG-1 Layer 3 (MP3)	.mp3	5623	240	128
5	Audio: WAV	.wav	27.619	160	1411

Tabla 4.2. Características de los ficheros de audio y video utilizados

Resultados:

A continuación se muestra una tabla con los valores obtenidos. Como se puede comprobar se ha realizado dos casos de estudio: con c*aching* habilitado y con *caching* deshabilitado.

Fichero	Tiempo de respuesta (seg) (*caching* habilitado)	Tiempo de respuesta (seg) (*caching* deshabilitado)
1	1,933	**1,272**
2	4,416	**2,113**
3	16,694	**4,786**
4	1,071	**0,941**
5	0,922	**0,792**

Tabla 4.3. Tiempos de respuesta para la reproducción (HTTP)

Era previsible que los datos que se obtuviesen cuando el *caching* estuviese deshabilitado fueran a ser menores que en el caso de estar habilitado.

Teniendo habilitada la opción de *caching*, la diferencia se acentúa cuanto mayor sea el tamaño del fichero a almacenar. A esto hay que sumar la diferencia de tratamiento de los datos en función del *codec* utilizado. Los *codecs* implementados por la librería JMF permiten el inicio de la reproducción antes de la descarga completa del fichero, como ocurre con los formatos MPEG y WAV. Sin embargo, en los formatos manejados por *codecs* del *plug-in* de Fobs, como DIVX, se ha de descargar el fichero completo previamente.

En el caso de deshabilitar el proceso de *caching*, las variaciones entre los valores obtenidos están directamente relacionadas con la tasa media de bits del fichero y la calidad que proporciona el formato de codificación. Así es como, por ejemplo, se pude explicar que con el fichero **2** se obtenga un valor inferior al **3** utilizando el mismo *codec*. En el primer caso se tiene una tasa de bits considerablemente superior haciendo que en poco tiempo el buffer del *codec* posea la cantidad

suficiente de datos para iniciar la decodificación. Un ejemplo del segundo racionamiento sería la diferencia entre los dos primeros. MPEG-1 es un formato de menor calidad que DIVX (basado en MPEG-4) por lo que su decodificación resulta menos costosa. A esto hay que añadir el factor de la eficiencia del *plug-in* que implementa el *codec*, con la consecuente influencia en el resultado.

En cuanto a los ficheros de audio, su marcada diferencia con los que contienen video es como consecuencia de las características propias de la codificación de audio, cuyo proceso está basado en algoritmos más simples. Entre los dos formatos analizados la diferencia entre ambos es debida principalmente a que en el formato WAV, al contrario que en el MP3, los datos no han sido comprimidos, por lo que en su decodificación no se ha de realizar un proceso de descompresión de los mismos.

Resumiendo, los valores de los tiempos de respuesta de la aplicación relativos al comienzo de la reproducción son dependientes de las características del fichero (tasa de bists, *codec*,...) para las dos configuraciones estudiadas, con *caching* habilitado y con *caching* deshabilitado. La desventaja clara de la primera sobre la segunda se suple con un uso más eficiente de la red y una mejor reproducción. En otras palabras, la reproducción del medio no se verá afectada como consecuencia de los posibles problemas derivados del uso de HTTP como protocolo *streaming* (caso de *caching* deshabilitado), principalmente asociados a la ocupación de la red.

2. Contenido enviado mediante flujos RTP:

Como ya se dijo en el apartado “Herramientas utilizadas” del capítulo anterior, no se encontró ninguna implementación de un *UPnP MediaServer* que proporcionase URLs para flujos RTP, por lo que se decidió adaptar el código de HomeAV para la prueba y así poder utilizar un Servidor RTP de los disponibles en el mercado. En concreto, para esta prueba se hizo uso de la aplicación JMStudio, distribuida junto con la librería JMF.

La adaptación del código de la aplicación consistió simplemente en no utilizar la URI que se recibe como argumento de entrada de la acción `SetAVTransportURI()` de la implementación del servicio *AVTransport Service*, y utilizar una definida específicamente para representar una sesión RTP. Ésta es del tipo que se definió en el apartado sobre la especificación de JMF para RTP del capítulo “Marco teórico” (Ejemplo: *rtp://224.0.0.1:1238/video/1*).

Se utilizó el mismo fichero de video MPEG anterior (**1**), el cual fue transcodificado por la propia aplicación JMStudio al formato definido en cada muestra.

Estos formatos son comúnmente utilizados en aplicaciones que envían y/o reciben flujos RTP, como aplicaciones de videoconferencia. Como ya se dijo en el marco teórico, RTP está diseñado para obtener un mejor comportamiento en la transmisión de flujos multimedia bajo condiciones de ocupación de red en las que HTTP no resulta efectivo por sus características.

Los valores se medirán habiendo comenzado la transmisión de los flujos previamente a la pulsación del botón “*Play*” de Cidero, para no considerar los retardos introducidos por el servidor en el inicio de su envío.

Resultados:

Los datos obtenidos se resumen en la siguiente tabla, en la que se especifica también el formato RTP del contenido transmitido.

Formato RTP	Tiempo de respuesta (seg)
Video: **MPEG** (320x176 píxeles) Audio: **MPEG** (44100 Hz)	**1,592**
Video: **JPEG** (320x176 píxeles) Audio: **MPEG** (44100 Hz)	**1,242**

Video: **H263** (320x176 píxeles) Audio: **G.723** (8000 Hz)	**1,272**
Audio: **GSM** (8000 Hz)	**1,061**
Audio: **MPEG** (44100 Hz)	**1,098**
Audio: **G.723** (8000 Hz)	**1,052**
Audio: **DVI** (8000 Hz)	**1,082**

Tabla 4.4. Tiempos de respuesta para la reproducción (RTP)

Observando la tabla se comprueba que los resultados obtenidos no difieren en exceso unos de otros. Sí hay una marcada diferencia entre los valores relativos a un flujo único de audio y los relativos a dos flujos, uno de video y otro de audio, que como se dijo en el análisis anterior, es consecuencia de la mayor complejidad del proceso de decodificación del video que retrasa el inicio de la reproducción.

En el caso de los flujos de video, la diferencia obtenida al usar MPEG es consecuencia de un proceso más complejo de decodificación que en el caso de JPEG. H263, a pesar de que su implementación está basada en un algoritmo aún más complejo, consigue un valor bajo ya que fue específicamente diseñado para RTP, permitiendo que se inicie antes la reproducción.

El resto de valores son tan próximos que no se puede establecer una relación directa entre estos y los formatos o procedimientos de decodificación utilizados.

En conclusión, la aplicación en poco más de un segundo desde la pulsación del "*Play*" en el *Control Point* puede iniciar la reproducción de un flujo de audio y/o video RTP.

5. Conclusiones

En este capítulo se recogen las conclusiones obtenidas de la realización de este proyecto.

En primer lugar se detalla el grado de cumplimiento de los objetivos planteados. Tras esto, se exponen las limitaciones que surgieron en el desarrollo del prototipo y, por último, se enuncian las posibles líneas futuras de trabajo para este tipo de dispositivos.

5.1 Conclusiones

Como objetivo general de este proyecto se marcó el desarrollo de un prototipo *UPnP MediaRenderer* que se integrase en plataformas OSGi.

En el apartado sobre el estado de arte del capítulo “Marco teórico”, se analizaron las actuales soluciones *UPnP MediaRenderer*. A partir del contenido de este análisis se concluye que no existe una solución que reuna todas las capacidades que se definieron como requisitos para el prototipo desarrollado.

El resultado es una aplicación Java para plataformas Windows y Linux, que permite la renderización de contenido multimedia (audio, video e imágenes), completamente conforme al estándar UPnP para este tipo de dispositivos. Gracias a esto, puede ser controlada remotamente por medio de acciones, entre las que se incluyen las más comunes de un reproductor. Además, es compatible con IPv6, y se integra en un *bundle* OSGi para su utilización en plataformas de servicios basadas en esta tecnología.

Por medio del uso de *Java Media Framework* (JMF), se consiguieron las capacidades para la reproducción de contenidos multimedia obtenidos a partir de flujos HTTP o RTP, con los formatos más comunes. A sí mismo, gracias a los paquetes Java AWT y Swing, se pudo dar soporte a la visualización de imágenes, también en este caso, con los formatos más comunes.

A continuación se analiza el cumplimiento de los subobjetivos específicos del proyecto, expuestos en la introducción del mismo:

1. *Estudio de las tecnologías implicadas para el diseño del prototipo:* se comenzó con la tecnología UPnP para la implementación de un protocolo de comunicación entre dispositivos, poniendo un mayor énfasis en aquellos aspectos más directamente relacionados con los dispositivos *UPnP MediaRenderer*s. También se presentaron los conceptos más interesantes de JMF como librería Java para el desarrollo de aplicaciones multimedia, para posteriormente dar una visión general de la Especificación OSGi que define las bases para integrar una aplicación en plataformas de servicios avanzados como las Pasarelas de Servicios. Por último, se analizó el estado del arte, no sólo del *UPnP MediaRenderer*, sino también por su interés, de la librería JMF presentando posibles alternativas que resuelvan sus limitaciones.

2. *Análisis de los requisitos del sistema en base a las características deseadas del prototipo:* en este sentido se definieron los requerimientos de la aplicación y a partir de estos se obtuvo el diagrama de Casos de Uso del sistema. Finalmente, se enumeraron las herramientas que han sido utilizadas en todo el proceso tanto de desarrollo como de pruebas del prototipo.

3. *Construcción de un prototipo de UPnP MediaRenderer:* tras haber adquirido los conocimientos teóricos necesario y analizar los requisitos, se paso al desarrollo del prototipo de dispositivo *UPnP MediaRenderer*. Se comenzó definiendo los diferentes módulos en los que se dividiría para posteriormente presentar las clases Java implementadas contenidas en el paquete que conforma cada módulo. Y para finalizar se incluyó un análisis de los diagramas de Secuencia más importantes del funcionamiento de la aplicación.

4. *Comprobación del correcto funcionamiento del prototipo en un escenario real y definición de las limitaciones derivadas de su diseño:* se realizaron varias pruebas en escenarios reales comprobando principalmente la correcta comunicación del dispositivo con el *Control Point* y el correcto procesamiento de la renderización. Se tomaron varias medidas de tiempos de respuesta de la aplicación para su valoración en esta memoria y así poder mostrar la eficiencia de la misma.

A nivel personal, opté por la realización de este proyecto por estar enfocado principalmente al entretenimiento digital en el hogar. Las tecnologías aplicadas a este tipo de ocio evolucionan a un ritmo vertiginoso y era la oportunidad de introducirme en el desarrollo de aplicaciones basadas en ellas. En este caso ha sido UPnP. Es fascinante la facilidad con la que un usuario puede disponer de una red de contenidos multimedia compartidos en la que se puede contralar su reproducción en dispositivos remotos desde un único dispositivo controlador.

A parte de esto, la implementación de la aplicación me ha servido para descubrir y aprender el manejo de herramientas realmente útiles para su desarrollo como es el caso de Eclipse, y adquirir nuevos conocimientos informáticos relacionados con la programación.

5.2 Limitaciones

Durante el desarrollo del proyecto se tuvieron que tomar decisiones y trabajar sobre problemas concretos surgidos de la integración de las distintas tecnologías en la aplicación. En consecuencia, se tuvieron que asumir ciertas limitaciones, que a continuación se enuncian:

- Publicación por una única interfaz: aunque la máquina sobre la que se ejecuta la aplicación tenga más de una interfaz activada, ésta se anunciará únicamente por una interfaz de red. Como se dijo, la IP que se utilizará para la comunicación se establece en el arranque del dispositivo, obtenida de la interfaz identificada por el parámetro `interfaceNumber` del archivo de configuración (Más información en “ANEXO IV”: Ficheros de la Aplicación).

- No soporte para listas de reproducción: la aplicación fue diseñada para la reproducción de ítems que conformasen un único fichero de audio, video o imagen, no permitiendo el manejo de listas de reproducción. En este sentido, las acciones del servicio *AVTransport Service*: `Next()` y `Previous()` se han implementado sin funcionalidad, y la acción `Seek()` se limita a la búsqueda de un punto temporal en un fichero, ya que su funcionalidad en la perspectiva de un *UPnP MediaRenderer* que no reproduce listas no resulta interesante.

- Manejo de un único ítem al mismo tiempo: el prototipo obtenido no implementa las acciones opcionales `PrepareForConnection()` ni `ConnectionComplete()` del correspondiente servicio *ConnectionManager Service*, no pudiendo mantener más de una conexión simultáneamente que le permitiese manejar varios ítems al mismo tiempo.

- Librerías Nativas (Linux): en el caso de plataformas Linux las librerías nativas de JMF no pueden ser empaquetadas en el “.jar” del *bundle*, ya que, al contrario que en Windows, no pueden ser accedidas en tiempo de ejecución en esa localización. En este caso, se hace necesario copiar el directorio con estas librerías en una ruta que posteriormente pueda ser añadida a través de la exportación de la variable del sistema *LD_LIBRARY_PATH* (Más información en “ANEXO V: Instrucciones de instalación”).

- Consumo de CPU: bien es sabido que las aplicaciones multimedia consumen gran cantidad de recursos de procesamiento. En el caso de aplicaciones programadas en Java, como en este caso, los requerimientos de estos recursos son mayores, principalmente como consecuencia del uso de la Máquina Virtual Java. Estos requerimientos hacen inviable trasladar la aplicación a dispositivos limitados como dispositivos móviles, al no poseer JVM, sino CVM o KVM las cuales no dan soporte a aplicaciones J2SE.

5.3 Líneas futuras de trabajo

En cuanto a las posibles mejoras que se podrían realizar sobre el prototipo, éstas estarían principalmente encaminadas a la resolución de algunas de las limitaciones expuestas anteriormente, aunque también se podrían añadir más. Ejemplo de estas futuras líneas de trabajo serían:

1. Permitir que el dispositivo se anunciase por todas las interfaces de red habilitadas de la máquina en la que se ejecute. De esta forma podría se descubierto por cualquier *Control Point* conectado a cualquiera de las subredes a las que tiene acceso. Posiblemente, esta capacidad implicaría también la utilización de *Control Point* capaces de discernir si la URL del documento de descripción del dispositivo que recibe en los paquetes SSDP *multicast*, es alcanzable o no en base a la IP que contiene.

2. Permitir establecer múltiples conexiones para la reproducción de varios ítems al mismo tiempo. Esto implicaría la implementación de los mencionados método `PrepareForConnection()` y `ConnectionComplete()` del servicio *ConnectionManager Service*, además de crear la lógica necesaria en las respectivas clases que implementan los servicios *AVTransport Service* y *RenderingControl Service* para poder manejar varias instancias de los mismos.

3. Añadir funcionalidad para el manejo de listas de reproducción. En este caso, las acciones `Next()` y `Previous()` si que serían útiles, y la acción `Seek()` podría asumir mayores capacidades, como saltar varias pistas adelante o detrás dentro de la lista de reproducción.

4. Añadir funcionalidad para *streaming* RTSP (*Real Time Streaming Protocol*) conforme a la especificación de la arquitectura UPnP para audio y video.

5. Añadir o crear nuevos *plug-ins* de *codecs* que puedan ser utilizados por la aplicación para la reproducción de nuevos formatos.

6. Estudiar nuevas alternativas a JMF que resuelvan las limitaciones derivadas de ésta librería, principalmente en cuestión de consumo de recursos del sistema, soporte de formatos de codificación o restricciones en su desarrollo por no estar sujeta licencias de código abierto. Esto podría implicar el uso de técnicas para el procesado multimedia programadas en otro lenguaje.

7. Importar el prototipo de *UPnP MediaRenderer* a dispositivos limitados como PDAs, SmartPhones, PocketPCs, etc. Para ello se podría hacer uso de plataformas como J2ME, para el desarrollo de aplicaciones Java en este tipo de dispositivos.

8. Incorporar políticas de seguridad mediante la implementación de nuevos servicios UPnP como el definido por el Foro UPnP *DeviceSecurity Service* [UPNPSECURITY].

9. Incorporar políticas de calidad de servicio mediante la implementación de servicios UPnP. Para ello, el Foro UPnP especifica una arquitectura para le gestión de calidad de servicio [UPNPQOS].

ANEXO I: Cidero/Cyberlink

El contenido de este anexo sirve de guía para la implementación de un dispositivo *UPnP MediaRenderer* mediante la utilización de la librería de clases Java de la aplicación Cidero.

1. Introducción

Cidero es una implementación de código abierto, del año 2004, basada en la librería Cyberlink, que integra varias aplicaciones conformes a UPnP tales como un *AV Control Point*, un *UPnP MediaServer* para acceder a estaciones de radio en Internet, y un *UPnP Bridge* para poder utilizar dispositivos No-UPnP, principalmente reproductores multimedia, dentro de la red UPnP.

Su código fuente contiene métodos y clases que hacen, ayudándose de Cyberlink, más sencilla la implementación de los servicios UPnP asociados a un *MediaRenderer* o un *MediaServer* que crearlos a partir de la propia librería Cyberlink.

Por tanto, en este proyecto se ha utilizado Cidero para múltiples usos:

- Como aplicación *AV Control Point* en el escenario de trabajo y pruebas.
- La librería Cyberlink que contiene para el desarrollo del dispositivo UPnP.
- Métodos y clases de sus propios paquetes para la implementación de los servicios asociados al *UPnP MediaRenderer*.

2. Implementación del dispositivo con Cyberlink

Se trata de un catálogo de métodos, clases e interfaces para la creación de dispositivos y Puntos de Control UPnP. Soporta todos los protocolos y mecanismos que define la especificación. Forma parte del dominio personal Cybergarage creado por Satoshi Konno en el año 2002, aunque desde entonces el código ha sido corregido y actualizado en varias ocasiones. La versión utilizada en este proyecto es la que se distribuye con Cidero, la 1.3.2.

Apoyado en la compatibilidad de Java, soporta IPv4 e IPv6 por defecto. Para poder seleccionar alguna de las versiones del protocolo IP a utilizar por defecto se ha de invocar al siguiente método:

`UPnP.setEnable(UPnP.USE_ONLY_IPV6_ADDR)`, para IPv6
o
`UPnP.setEnable(UPnP.USE_ONLY_IPV4_ADDR)`, para IPv4

En el caso de IPv6 se hace necesario definir el alcance de la red donde se va a agregar el dispositivo o Punto de Control, ya que el direccionamiento *multicast* depende de ello. Esto afecta directamente a la dirección IPv6 *multicast* a utilizar en SSDP. Por defecto el alcance que define Cyberlink es local. Para modificarlo se usa el siguiente método:

```
UPnP.setEnable(UPnP.USE_IPV6_SITE_LOCAL_SCOPE)
```

(*Site Local*: alcance para una red doméstica)

A continuación se comentarán aquellos aspectos de la lógica de la librería que resultan interesantes con este proyecto y que están orientados a crear un dispositivo UPnP.

Paquetes de la librería

- `org.cybergarage.http`: para la creación y configuración de los servidores HTTP necesarios en la comunicación, así como los mensajes tipo de solicitud y respuesta HTTP, y toda la funcionalidad asociada a este protocolo.

- `org.cybergarage.net`: para la configuración de los parámetros de red.
- `org.cybergarage.soap`: para la implementación del protocolo SOAP para la invocación remota de acciones.
- `org.cybergarage.upnp` (ssdp, control, eventos, dispositivos y servicios): cubre toda la lógica para la creación de puntos de control, dispositivos y servicios, además de la implementación de los mecanismos de descubrimiento (SSDP), de control (SOAP) y eventos (GENA) que especifica UPnP.
- `org.cybergarage.util`: contiene un conjunto de clases e interfaces que son utilizadas como herramientas para la implementación del código del resto de paquetes.
- `org.cybergarage.xml`: para dar soporte al tratamiento, análisis y generación de documentos en formato XML que se utilizan para la creación de mensajes en varios de los protocolos UPnP, así como los descriptores de dispositivos y servicios.

Descripción de la funcionalidad

Descripción del dispositivo y servicios contenidos

El paso previo a la implementación del dispositivo es la creación de los documentos XML descriptores tanto del dispositivo como de los servicios que contiene. Estos ficheros serán cargados al inicio de la comunicación para estar disponibles en la petición de los mismos por parte del Punto de Control durante el proceso de descubrimiento.

El campo *URLBase* del descriptor del dispositivo no debe figurar en este ya que es generado automáticamente cuando se recibe la petición del Punto de Control para obtener este documento.

Inicio y abandono

Para el inicio de la comunicación, la clase `Device` define el método `start()`. La invocación de este método desencadena la creación de un servidor HTTP por interfaz, para poder recibir los mensajes de los distintos protocolos UPnP (SOAP, GENA y HTTP GET para la obtención de los descriptores) y dos *sockets*, uno *unicast* y otro *multicast*, para el protocolo SSDP. Esto supone que desde este momento coexisten varios hilos de ejecución para el envío de peticiones, la recepción de los mensajes y la generación de las correspondientes respuestas conforme a lo especificado para cada protocolo mencionado.

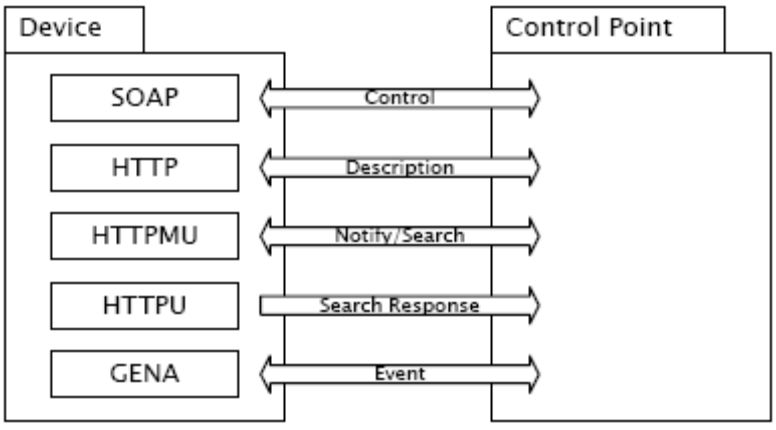


Figura I.1. Servidores HTTP del dispositivo

Este proceso conlleva la carga de los descriptores del dispositivo y servicios para estar disponibles para su obtención por el Punto de Control.

Los parámetros mostrados por la siguiente tabla son configurables por medio de métodos de `Device`:

Parámetro	Valor por defecto	Método	Descripción
`HTTP port`	4004	`setHTTPPort()`	Puerto para abrir la conexión del servidor HTTP
`Description URI`	/description.xml	`setDescriptionURI()`	URI, localizador, donde se encuentra el descriptor del dispositivo
`Lease time`	1800	`setLeaseTime()`	Tiempo en segundos durante el cual un anuncio SSDP es válido. Pasado éste y si no se ha recibido otro mensaje, el Punto de Control entiende que el dispositivo no está disponible

Tabla I.1. Parámetros configurables de la clase `Device`

Cada vez que se recibe una solicitud HTTP enviada por el Punto de Control, se clasifica según sea una solicitud HTTP GET para la obtención del descriptor, una solicitud SOAP para la invocación de una acción de un servicio o una solicitud de Subscripción al mecanismo de eventos (GENA) de algún servicio. Esto se realiza en el método `httpRequestReceived()` de `Device`.

Notificación

Tras esto, el método continúa su ejecución realizando el proceso de anuncio enviando mensajes ALIVE, siguiendo las pautas que dicta el protocolo SSDP. Se anuncia tanto para el dispositivo, como los dispositivos embebidos y servicios que contiene.

Para el abandono de la red UPnP, se ha de invocar al método *`stop()`* de la misma clase. Su invocación provoca el envío de la correspondiente notificación al Punto de Control con un mensaje BYEBYE acorde al protocolo SSDP. Igualmente que el anuncio, se notifica para el dispositivo así como para los dispositivos embebidos y servicios que contiene.

Dispositivos embebidos

Como ya se mencionó en el apartado del capítulo “Marco Teórico” que hacía referencia a UPnP, un dispositivo podía contener otros dispositivos embebidos. Para obtener la lista de dispositivos embebidos la clase `Device` proporciona el método `getDeviceList()`. Se puede buscar un dispositivo embebido por nombre o por UDN (*Unique Device Name*), usando el método `getDevice()` de la misma clase.

Servicio

Para obtener una lista de los servicios contenidos por el dispositivo se utiliza el método `getServiceList()` de la clase `Device`. De cada servicio se puede obtener la lista de variables de estado y acciones asociadas. Se usa el método `getServiceStateTable()` para el primer caso, y `getActionList()` para el segundo. Ambos métodos definidos en la clase `Device`.

Se puede buscar un servicio por medio de su identificador, invocando el método `getService()`. Y se puede buscar una acción o una variable de estado de un servicio por su nombre invocando `getAction()` y `getStateVariable()` respectivamente. Todos estos métodos se encuentran en la clase `Device`.

Control

Para enterarse de cuándo se ha recibido una solicitud de invocación de una acción, se ha de implementar la interfaz *`ActionListener`*. Este escuchador debe implementar el método

`actionControlReceived()` de esta interfaz. Este método contiene la acción que se desea invocar y un parámetro con la lista de argumentos de entrada. Estos argumentos tienen los valores que el Punto de Control establece.

En respuesta, se establecen los argumentos de salida de la acción y se devuelve "true" cuanto la petición resulta exitosa. Sin embargo, si no resulta así, se devuelve el valor *false* lo que provoca que se devuelva una respuesta *UPnPError* al Punto de Control. Esta respuesta automática consta del mensaje *INVALID_ACTION* que define SOAP. Además, se utiliza el método `setStatus()` de la clase `StateVariable` para devolver otros errores UPnP.

También Cyberlink da soporte para la implementación del mecanismo de control, *Query for variable*, para preguntar el valor de una variable de estado de un servicio, pero como se dijo en el apartado del marco teórico que habla sobre UPnP, éste mecanismo fue deprecado por el Foro UPnP. Si se desea averiguar que métodos y clases define Cyberlink para este mecanismo, se puede consultar en la guía de programación de esta librería que se encuentra disponible en la web de Cybergarage [CYBERGARAGE].

Se usa el método `setActionListener()` de `Action` para añadirse como escuchador de la acción.

Eventos

El Punto de Control puede subscribirse para la recepción de eventos del dispositivo. Es el dispositivo y no el Punto de Control quien maneja los mensajes de subscripción automáticamente. Cuando dispositivo recibe un mensaje de subscripción de un Punto de Control, éste es incluido por el dispositivo en la lista su lista de subscriptores, mientras que si el mensaje es para borrar una subscripción es borrado de esta lista. A su vez, el dispositivo también maneja los mensajes que recibe periódicamente para renovar esta subscripción, tal y como especifica el estándar UPnP para este mecanismo de eventos.

Para que el dispositivo establezca o actualice el valor de una variable de estado, se debe utilizar el método `setValue()` de la clase `StateVariable`. Si las modificaciones de esta variable de estado deben ser informadas mediante este mecanismo de eventos, se enviará el correspondiente evento a los subscriptores.

Es importante comentar que en Ciberlink, la lista de dispositivos embebidos, la lista servicios asociados, la lista de acciones y la lista de variables de estado de estos servicios, se generan a partir de los documentos descriptores del dispositivo y servicios del mismo.

En la siguiente figura se muestra el Diagrama de Clases de Cyberlink para la creación de dispositivos UPnP.

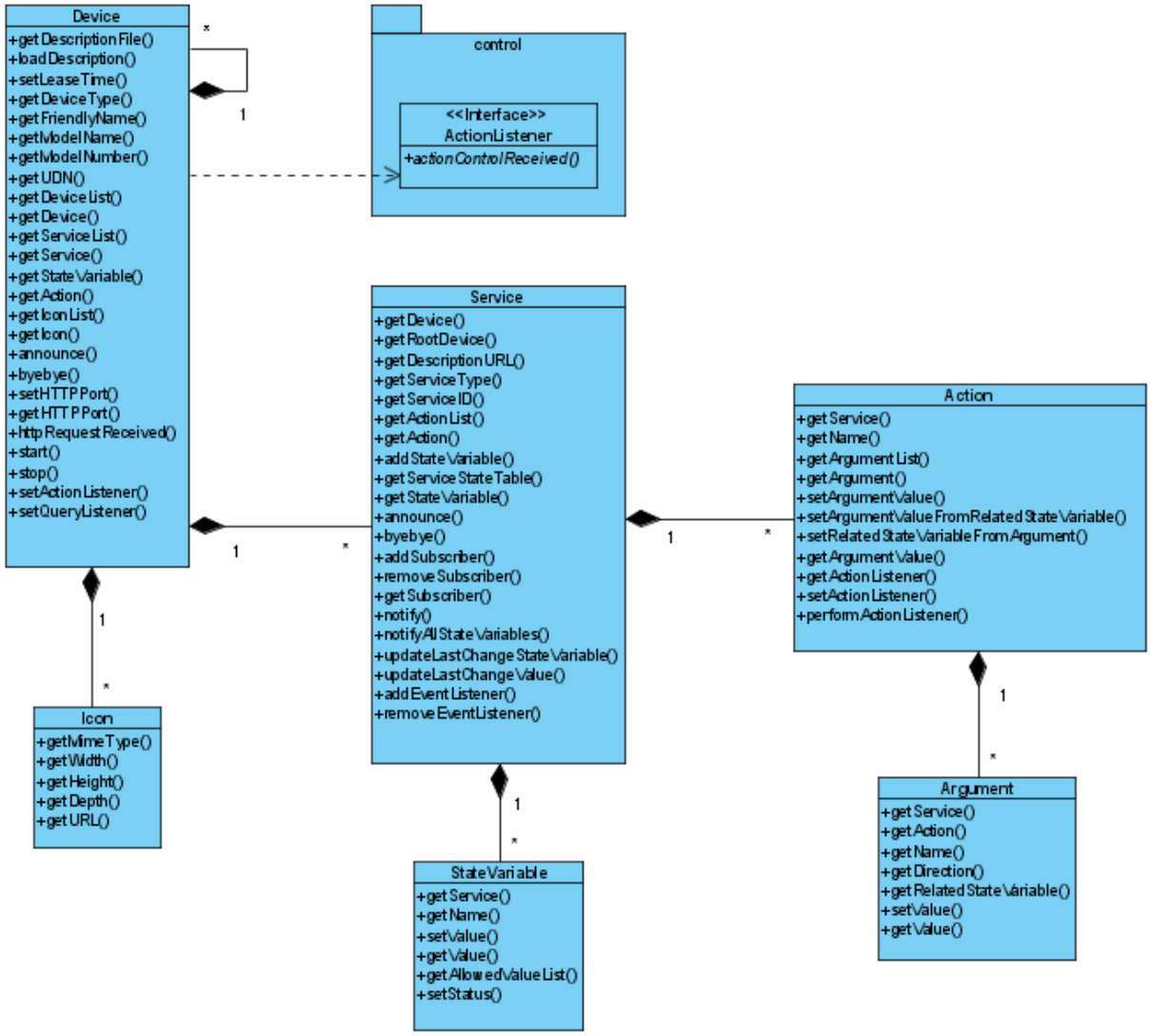


Figura I.2. Diagrama de Clases de Cyberlink para dispositivos

Entre la clase `Device` y la interfaz *`ActionListener`* existe una relación de dependencia. Se trata de una relación débil, más incluso que una Asociación, establecida simplemente porque la primera hace uso de un objeto de la segunda. En este caso `Device` hace uso de *`ActionListener`* para añadir un escuchador, que quiere enterarse de cuando se recibe una acción, a la lista de escuchadores que tiene cada acción de cada servicio.

El resto de relaciones son de composición, es decir, que un objeto de la clase base está constituida por uno o varios objetos de la otra. El tiempo de vida de estos objetos está condicionado al tiempo de vida del que los incluye. Como ejemplo, un dispositivo (*Device*) está compuesto por 1 o más dispositivos, y por uno o más servicios (*Service*), y estos dispositivos embebidos y servicios asociados sólo existen mientras exista el dispositivo que los contiene.

Para más información sobre el uso de la librería Cyberlink para la implementación de un Punto de Control se recomienda consultar la guía de programación de Cyberlink [CYBERGARAGE].

3. Implementación de los servicios con Cidero

Como ya se ha dicho, de entre el conjunto métodos y clases que define Cidero, en base a Cyberlink, existen cuatro clases que resultan realmente útiles para implementar los servicios asociados a un *UPnP MediaRenderer* y un *UPnP MediaServer*.

Puesto que este proyecto versa sobre el diseño de un *MediaRenderer*, se verá el caso para la implementación de los servicios asociados a este tipo de dispositivo UPnP.

El paquete `com.cidero.upnp` de Cidero contiene el conjunto de clases e interfaces relacionadas con la funcionalidad UPnP. En él figuran las cuatro clases necesarias para definir los servicios UPnP del *MediaRenderer*: `RenderingControl`, `ConectionManager` y `AVTransport` que extienden a la clase abstracta `AbstractService`.

Creación de un servicio

`AbstractService` es la clase base para toda clase que implemente un servicio específico en Cidero. Extiende la funcionalidad de la clase Service de la librería CiberLink.

Desde la perspectiva de un dispositivo, esta clase permite un mapeo automático entre la llegada de la invocación de una acción UPnP y un método con el mismo nombre de esa acción definido en la clase derivada. Para ello se utiliza el método `actionControlReceived()` de A`bastractService`, en el que se procesa esta invocación y se llama al correspondiente método.

Implementación de los servicios para un UPnP MediaRenderer

Las clases `RenderingControl`, `ConnectionManage` y `AVTransport` son las clases base para la implementación de los servicios de un *UPnP MediaRenderer*. Definen, entre otros, los métodos que implementan las acciones con el mismo nombre que la acción implementada. Algunos de estos métodos se definen sin funcionalidad, únicamente devuelven un *false* para indicar que esa acción no es soportada. De esta forma, el programador que quiera implementar esa acción deberá crear sus propias clases derivadas de éstas e implementar dichas acciones sobrescribiendo estos métodos.

Para las acciones soportadas, sus respectivos métodos manejan los argumentos de entrada y salida de la acción, así como actualizan los valores de las variables de estado que se vean modificadas por ellos. Las clases derivadas deberán considerar ésta lógica para las implementaciones de las que acciones que no están implementadas en la clase base y lo realicen del mismo modo, por coherencia con el conjunto de métodos y para cumplir con la especificación.

El formato que tiene el nombre de estos métodos es:

`action<Nombre de la acción>()`

Cuando se crea una instancia de un servicio específico se utiliza el método `initializeStateVariables()`, sobrescrito en la clase derivada de `AbstractService`, para inicializar los valores de las variables de estado del servicio correspondiente.

En la siguiente figura se muestra el Diagrama de Clases de Cidero con las relaciones entre las clases mencionadas. De entre los métodos mostrados de las clases `RenderingControl`, `ConnectionManager` y `AVTransport`, se pueden observar aquellos que implementan las acciones.

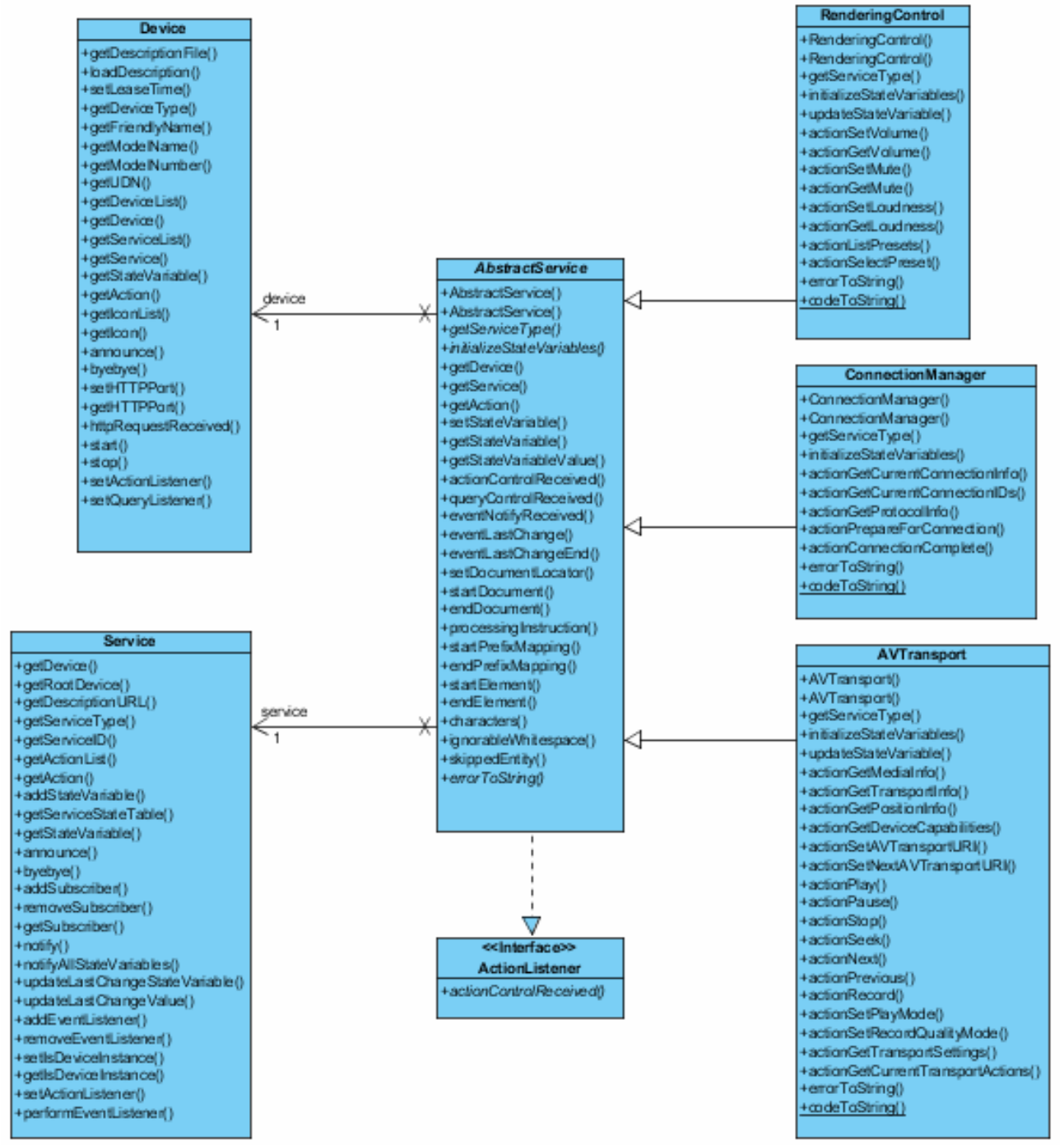


Figura I.3. Diagrama de Clases de Cidero para servicios de un *UPnP MediaRenderer*

La relación que se define entre la clase abstracta `AbstractService` y las clases `Service` y `Device` es una Asociación. Estos es, un objeto de `AbstractService` tiene asociado un objeto `Device` y un objeto `Service`. Pero en este caso, el ciclo de vida de los objetos asociados es independiente del ciclo de vida del objeto que los usa. En el primer caso esta asociación es utilizada para obtener el dispositivo al que está asociado el servicio en concreto y en el segundo lugar permite el uso de métodos de la clase `Service` necesarios para implementar la funcionalidad de este servicio.

`AbstractService` implementa la interfaz *`ActionListener`*. En la creación de una instancia de la clase, ésta se añade a la lista de escuchadores de cada acción del servicio que la extiende invocando directamente el método `setActionListener()` de `Action`.

ANEXO II: Modificaciones en Cidero

En las próximas líneas se recogen aquellas modificaciones y adaptaciones, sobre las clases de la librería proporcionada por la aplicación Cidero, que fueron necesarias para el correcto funcionamiento del prototipo de *UPnP MediaRenderer* diseñado.

1. Introducción

Como ya se ha dicho en varias ocasiones en la implementación de HomeAV se ha hecho uso del catálogo de métodos y clases Java que provee la aplicación Cidero.

Durante el proceso de pruebas realizadas para evaluar la comunicación entre HomeAV (*UPnP MediaRenderer*) y el *Control Point*, se han ido obteniendo distintos errores que han obligado a modificar algunos del los métodos de estas clases.

Las modificaciones realizadas buscan dos objetivos:

1. El correcto desarrollo de la comunicación UPnP tanto en redes IPv4 como IPv6.

2. La adaptación de las clases a las condiciones que se derivan de la integración del *UPnP MediaRenderer* en un *bundle* OSGi.

2. Primer objetivo: Correcto desarrollo de la comunicación UPnP

Para alcanzar este objetivo se tuvo que modificar la lógica para la obtención de la dirección IP local donde se ejecuta la aplicación que se entiende como la IP del dispositivo (HomeAV) usada en la comunicación.

El principal escoyo que se encontró fue durante las pruebas de la aplicación, cuando el equipo tenía activas varias interfaces de red de las que únicamente una conectaba con la subred en la que se encontraba el *Control Point*. La lógica implementada para el descubrimiento le permitía a la aplicación enviar paquetes SSDP *multicast* por todas las interfaces activas y conteniendo en cada paquete la URL para obtener el descriptor del dispositivo formada con la IP de la interfaz correspondiente.

En este caso, y tanto con IPv4 como IPv6, surgía un error en Cidero cuando recibía estos anuncios SSDP, ya que seleccionaba el último en llegar para obtener la URL que le permitiese acceder al documento descriptor, sin tener en cuenta si se trataba de una URL formada con una IP perteneciente a su subred u otra IP perteneciente a otras de las subredes a las que el *MediaRenderer* estaba conectado

Además de esto, surgieron más problemas cuando la comunicación se hacía en escenarios IPv6, como se describe a continuación:

- En primer lugar, uno de los fallos detectados deriva de la propia J2SE, al no entender que la dirección de la interfaz loopback “fe80::1” es precisamente de este tipo, permitiendo la utilización de la misma para la configuración de la comunicación del dispositivo en su inicio. Esto se comprobó en el método `getHostAddress()` de la clase `org.cybergarage.net.HostInterface`, cuando se hace uso del método `isLoopbackAddress()` de `java.net.InetAddress`.

- En segundo lugar, el problema surgió cuando el dispositivo utilizaba direcciones válidas IPv6 para enlace local (en Inglés: *link local*) en las que se añade tras la propia dirección un identificador numérico de la interfaz a la que está asociada y que resulta necesario para su uso. En este caso, el error aparece al no tener, dispositivo y *Control Point*, el mismo identificador de red. La URL del descriptor del dispositivo contiene una identificador que corresponde con la interfaz del *UPnP MediaRenderer* por la que se envió el paquete SSDP *multicast*, Cidero no comprueba este identificador, y mucho menos lo cambia por el suyo, por lo que no puede recuperar el documento.

Por todo ello, se decidió que el dispositivo se anunciase por una sola interfaz, formándose una única URL para recuperar su documento descriptor. Como se verá en la descripción de la primera modificación, la IP utilizada sería pues con la que se configurasen los *sockets* SSDP y servidores HTTP conformes al protocolo UPnP. Por tanto, toda la comunicación, bajo la perspectiva de este dispositivo (HomeAV), va a estar basada en la utilización de dicha IP.

Modificación 1

Se modificó el método que provee Cyberlink para la obtención de las IPs locales **`getHostAddress()`** de la clase **`org.cybergarage.net.HostInterface`**. Lo único que se hizo fue añadir el código que quita el identificador de interfaz IPv6 de la dirección IPv6 obtenida. Se hizo necesario esto en el caso en el que se ejecutaba el código en la plataforma Linux, ya que se añade este identificador y debe ser quitado para que el resto de métodos de la aplicación que usen esta IP la puedan reconocer como válida.

Con la dirección IP obtenida de este método se crea:

- El servidor HTTP utilizado para la implementación de los protocolos SOAP para el control, GENA para los eventos, y HTTP GET para la obtención del descriptor del dispositivo.

- El *socket* para la recepción de los mensajes SSDP de búsqueda de dispositivos que manda el *Control Point*.

Como se verá en las descripciones de las siguiente modificaciones, éste método va a ser invocado cada vez que se necesite obtener la IP local del dispositivo que está usando en la comunicación, en vez del método `getLocalAddress()` de `java.net.Socket` con el que se podrían producir errores derivados de lo expuesto anteriormente (principalmente que obtenga una IP de loopback como la mencionada “fe80::1”).

En el siguiente cuadro de texto se muestra el código de este método ya modificado.

```
public final static String getHostAddress(int n){
	if (hasAssignedInterface() == true){
		return getInterface();
	}
	int hostAddrCnt = 0;
	try {
		Enumeration nis = NetworkInterface.getNetworkInterfaces();
		while (nis.hasMoreElements()){
			NetworkInterface ni =
			(NetworkInterface)nis.nextElement();
			Enumeration addrs = ni.getInetAddresses();
			while (addrs.hasMoreElements()) {
				InetAddress addr =
				(InetAddress)addrs.nextElement();
				if (isUsableAddress(addr) == false)
					continue;
				if (hostAddrCnt < n) {
					hostAddrCnt++;
					continue;
				}
				String host = addr.getHostAddress();
				if(isIPv6Address(host)){
				if(isGlobalScopeAddress ((Inet6Address)addr)){
						int index=host.indexOf('%');
						if(index > 0){
							host=host.substring(0,
							index);
						}
					}

				}
				return host;
			}
		}
	}
	catch(Exception e){};
	return "";
}
```

Cuadro de texto II.1. Método `getHostAddress()` de `org.cybergarage.net.HostInterface`

Modificación 2

Se actualizó el código de la clase **`Device`** del paquete **`org.cybergarage.upnp`** a la versión que se incluye en la aplicación CMGate del mismo creador que la librería. Esta versión corrige algunos fallos de la anterior y además añade mayor funcionalidad. Lo más interesante para HomeAV y por tanto lo que indujo a hacer este cambio fue la modificación de los métodos `announce()` y `byebye()` utilizados en la fase de descubrimiento (que utiliza el protocolo SSDP), para que se cierren los *sockets* SSDP tras el envío de cada notificación.

Además, se modificó el código del método **`httpGetRequestReceived()`** en aquellas sentencias en las que se obtenía la dirección local del dispositivo (HomeAV) mediante un procedimiento que implicaba la invocación al mencionado método `getLocalAddress()` de `java.net.Socket`. Obviamente, se utiliza la llamada al método `getHostAddress()` de `HostInterface`, tal y como se advirtió anteriormente.

```
        private void httpGetRequestReceived( HTTPRequest httpReq )
        {
          /**This method is invoked when the Control Point sends a HTTP-GET
              request to obtain the Device description file or the Service
              description files
             */
                String uri = httpReq.getURI();
                if (uri == null) {
                       httpReq.returnBadRequest();
                       return;
                }
                Device embDev;
                Service embService;
                byte fileByte[] = new byte[0];
                if (isDescriptionURI(uri) == true) {
                       String localAddr = HostInterface.getHostAddress(0); //before
                       -> httpReq.getLocalAddress();
                       fileByte = getDescriptionData(localAddr);
                }else if ((embDev = getDeviceByDescriptionURI(uri)) != null) {
                       String localAddr = HostInterface.getHostAddress(0); //before
                       -> httpReq.getLocalAddress();
                       fileByte = embDev.getDescriptionData(localAddr);
                } else if ((embService = getServiceBySCPDURL(uri)) != null) {
                       fileByte = embService.getSCPDData();
                } else {
                       httpReq.returnBadRequest();
                       return;
                }
                HTTPResponse httpRes = new HTTPResponse();
                if (FileUtil.isXMLFileName(uri) == true)
                       httpRes.setContentType(XML.CONTENT_TYPE);
                httpRes.setStatusCode(HTTPStatus.OK);
                httpRes.setContent(fileByte);

                httpReq.post(httpRes);
        }
```

Cuadro de texto II.2. Método `httpGetRequesteReceived()` de `com.cybergarage.upnp.Device`

Modificación 3

Se modificó el método **`receive()`** de las clases **`HTTPUSocket`** y **`HTTPMUSocket`** del paquete **`org.cybergarage.upnp.ssdp`**. Se modifica la invocación a los respectivos métodos `setLocalAddress()` de ambas clases. Previamente se establecía la IP local de la forma que se viene diciendo, invocando al método `getLocalAddress()` de la clase `java.net.Socket`. Con esta modificación, la dirección local se establecerá a partir de la obtenida en la llamada al método `getHostAddress()` de `HostInterface`, como en el anterior caso.

Aunque el método `receive()` de `HTTPUSocket` no es invocado en la perspectiva del *MediaRenderer*, se decidió modificarlo también por coherencia entre ambas clases.

```
public SSDPPacket receive(){
	byte ssdvRecvBuf[] = new byte[SSDP.RECV_MESSAGE_BUFSIZE];
	SSDPPacket recvPacket = new SSDPPacket(ssdvRecvBuf,
	ssdvRecvBuf.length);
	recvPacket.setLocalAddress(HostInterface.getHostAddress(0));
	//before->(getLocalAddress());
	try {
		ssdpMultiSock.receive(recvPacket.getDatagramPacket());
		recvPacket.setTimeStamp(System.currentTimeMillis());
	}
	catch (Exception e) {
		//Debug.warning(e);
	}
	return recvPacket;
}
```

Cuadro de texto II.3. Método `receive()` de `org.cybergarage.upnp.ssdp.HTTPMUSocket`

Modificación 4

En este caso se modificó el método **`setHost()`** de la clase **`org.cybergarage.http.HTTPPacket`**. Únicamente se añadió un condicionante para el caso en que se obtengan direcciones IPv6 en un formato en el que la IP no está contenida entre dos corchetes (ejemplo: 2001::1). Esto ocurría con las direcciones IPv6 cuando se ejecutaba en el equipo Windows, y se hacen necesarios estos caracteres para formar correctamente el campo *HOST* de los mensajes HTTP. Con el nuevo código estos caracteres se añadirán.

```
public void setHost(String host, int port)
{
	String hostAddr = host;
	if (HostInterface.isIPv6Address(host) == true){
		if(!(host.startsWith("[") && host.endsWith("]")))
			hostAddr = "[" + host + "]";
	}
	setHeader(HTTP.HOST, hostAddr + ":" + Integer.toString(port));
}
```

Cuadro de texto II.4. Método `setHost()` de `org.cybergarage.http.HTTPPacket`

3. Segundo objetivo: Adaptación para la integración en un *bundle* OSGi

Como se comentó en el capítulo “Marco Teórico”, el *Framework* de OSGi provee de un contexto de ejecución propio al *bundle*. Para acceder a los recursos, como archivos, del *bundle* se ha de utilizar este contexto.

Los métodos definidos en Cidero y Cyberlink para recuperar estos descriptores se basan en la creación de un objeto `File` a partir de la ruta relativa del fichero. Sin embargo, esto no es aplicable cuando se integra la aplicación en un *bundle* por lo dicho anteriormente.

Sabiendo esto, se implementó en HomeAV la lógica necesaria para poder acceder a los documentos descriptores del dispositivo y servicios alojados en el *bundle*. Como resultado se utilizará el contexto del *bundle* para acceder a estos recursos, obteniendo la URL de cada descriptor. A partir de esta URL se abrirá una conexión para recuperar un flujo de entrada (`InputStream`) con los datos contenidos en cada fichero.

Por tanto, se dispondrá de un flujo de entrada por cada descriptor en vez de un objeto `File`. En consecuencia se tuvo que realizar algunas modificaciones en las clases relacionadas con el manejo de estos descriptores, entre las que se incluyen clases de Cidero y clases de la librería Ciberlink.

Modificación 5

Es la principal modificación realizada. Se trata de la creación del método **`parseInputStream()`** en la clase **`org.cybergarage.xml.Parser`** utilizada para la recuperación de la información de los documentos XML. Está basado en el método de la misma clase `parse(File descriptionFile)`. El flujo de entrada se analiza directamente y tras esto se cierra.

```
////////////////////////////////////////////
//     parse (InputStream)
////////////////////////////////////////////
public Node parseInputStream(InputStream inStream) throws
 ParserException{
       try{
              InputStream inputStrm = inStream;
              Node root = parse(inputStrm);
              inputStrm.close();
              return root;

       } catch (Exception e) {
              logger.warning("ERROR");
              return null;
       }

}
```

Cuadro de texto II.5. Método `parseInputStream(InputStream)` de `org.cybergarage.xml.Parser`

Modificación 6

Se añade el método **`loadDescription(InputStream descStream)`** a la clase **`org.cybergarage.upnp.Device`**. Se basa en el método análogo `loadDescriptionFile(File file)`. Permite cargar el descriptor del servicio a partir del flujo de entrada de su contenido. Utiliza el método visto anteriormente para recuperar la información del documento.

Este método es utilizado en el constructor de la clase `homeAV.upnp.HomeAVMediaRenderer` de HomeAV.

```
public boolean loadDescription(InputStream descStream)
throws InvalidDescriptionException {
       try {  Parser parser = UPnP.getXMLParser();
              rootNode = parser.parserInputStream(descStream);
              if (rootNode == null)
                     throw new InvalidDescriptionException(
                     Description.NOROOT_EXCEPTION);
              deviceNode = rootNode.getNode(Device.ELEM_NAME);
              if (deviceNode == null)
                     throw new InvalidDescriptionException(
                     Description.NOROOTDEVICE_EXCEPTION);
       } catch (ParserException e) {
              throw new InvalidDescriptionException(e);
       }if (initializeLoadedDescription() == false)
              return false;
       setDescriptionFile(null);
       return true;
}
```

Cuadro de texto II.6. Método `loadDescription(InputStream)` de `org.cybergarage.upnp.Device`

Modificación 7

Se modificó el método **`getSCPDNode()`** de **`org.cybergarage.upnp.Service`** utilizado, en varias clases de Ciberlink, para poder recuperar información del documento XML de descripción de un servicio. Con la modificación se busca poder obtener esta información a partir de un flujo de entrada que, como se comentó, es lo que obtenemos al acceder a un fichero descriptor contenido en el *bundle*. Para ello se tuvieron que crear varios métodos:

- **`getSCPDNode(InputStream inputStrm)`**
- **`getServiceDescriptionInputStream()`**
- **`setServiceDescriptionURL()`**

, además de una variable **`servDescrpURL`**.

Para que el código del método mantuviese su misma funcionalidad se hizo que se tratase la excepción del `Parser`, lanzado al no poder obtener el descriptor de la forma usual por no acceder al contexto del *bundle*, y se invocase al método `getSCPDNode(InputStream)` creado para este caso.

A continuación se muestra el código modificado del método `getSCPDNode()`:

```
try {
        scpdNode = getSCPDNode(new File(newScpdURLStr));;
}
catch (Exception e4) {
        if(e4 instanceof ParserException){
                try{//(OSGI)
                        scpdNode = getSCPDNode(
                        getServiceDescriptionInputStream();
                }
                catch(Exception e){
                        logger.warning("Impossible to parse the
                        Service description file");
                        Debug.warning(e);
                }
        }else
                Debug.warning(e4);
}
```

Cuadro de texto II.7. Código modificado del método `getSPDNode()` de `org.cybergarage.upnp.Service`

El primero de los métodos creados, **`getSCPDNode(InputStream input Strm)`**, es una variación de otro de la misma clase, `getSCPDNode(File file)`, para el caso en que se quiera obtener la información del descriptos a parte de un flujo de entrada. Para ello, hará uso del método ya comentado `parseInputStream()`.

```
private Node getSCPDNode(InputStream inputStrm) throws ParserException{
        Parser parser = UPnP.getXMLParser();
        return parser.parseInputStream(inputStrm);
}
```

Cuadro de texto II.8. Método `getSCPDNode(InputStream)` de `org.cybergarage.upnp.Service`

Los otros dos métodos añadidos son, **`setServiceDescriptionURL()`** y **`getServiceDescriptionInputStream()`**, y una variable, **`servDescrptURL`**. La variable guarda la URL del descriptor del servicio que corresponda y es utilizado en los dos métodos anteriores. En el primer caso para establecer el valor de esta variable, y en el segundo para abrir una conexión con esta URL y recuperar un flujo de entrada con los datos contenidos en el fichero.

El método `setServiceDescriptionURL()` es invocado y por tanto establecida la URL, en el constructor de la clase `com.cidero.upnp.AbstractService`, creado específicamente para ello y que del que se hablará en el siguiente apartado de modificaciones. Mientras, el otro método `getServiceDescriptionInputStream()`, como ya se ha visto, es utilizado en el método `getSCPDNode()`.

```
public void setServiceDescriptionURL(URL servDescrptURL){
        logger.info("Service description URL: " + servDescrptURL);
        this.servDescrptURL = servDescrptURL;
}
```

Cuadro de texto II.9. Método `setServiceDescriptionURL()` de `org.cybergarage.upnp.Service`

```
private InputStream getServiceDescriptionInputStream(){
        InputStream is = null;
        try {
                is = servDescrptURL.openStream();
        } catch (IOException e) {
                // TODO Auto-generated catch block
                logger.warning("An error occurs opening the stream");
        }
        return is;
}
```

Cuadro de texto II.10. Método `getServiceDescriptionURL()` de `org.cybergarage.upnp.Service`

Modificación 8

Las últimas modificaciones afectan a las clases de Cidero utilizadas por HomeAV.

En primer lugar, se creó un nuevo constructor en la clase abstracta **`com.cidero.upnp.AbstractService`**. Las diferencias respecto del otro constructor ya definido son:

1. Posee un argumento de entrada más, la URL del descriptor del servicio.
2. Invoca al método `setServiceDescriptorURL()`, visto anteriormente, para pasar esta URL y que así se pueda recuperar cuando se invoque el otro método, `getServiceDescriptorInputStream()`, de la clase `Service`.

```
	/**
	 * Constructor (OSGI). Setup all the action/query listeners for      *
	 this service,using the CLink device service with the same name as * this
	 service
	 * @param      CLink device object, Service description URL
	 * @exception  Throws InvalidDescriptionException if device has no
	 *             matching service
	 */
	public AbstractService( Device device, URL servDescrptURL )
	throws InvalidDescriptionException
	{
	// Get underlying Cybergarage service object for this service
	//and install actionControlReceived() 'dispatcher' method below for
	// all actions
		this.device = device;
		ServiceList serviceList = device.getServiceList();
		if (serviceList == null)
		{
			logger.severe("Null service list in service constructor");
			throw new InvalidDescriptionException("Null device service
			list");
		}
		int n;
		for (n = 0; n < serviceList.size(); n++)
		{
			Service service = serviceList.getService(n);
			logger.fine("ServiceType = "+service.getServiceType());
			logger.fine("ServiceId = " + service.getServiceID());
			if( service.isService(getServiceType()) )
			{
				service.setServiceDescriptionURL(servDescrptURL);

				logger.fine("Matching service found ");
				this.service = service;
				service.setQueryListener( this );
				service.addEventListener( this );
				ActionList actionList = service.getActionList();
				for (int j = 0; j < actionList.size(); j++)
				{
					Action action = actionList.getAction( j );
					action.setActionListener( this );
				}
				break;
			}else{
				logger.fine("Not a matching service ");
			}
		}
		if (n == serviceList.size())
		{
			throw new InvalidDescriptionException("Device doesn't
			support service '" + getServiceType() );
		}
		// If this service is being instantiated on the *device* side
		//)initialize state variables
		if( device.getInstancePerspective() == UPnP.DEVICE )
			initializeStateVariables();
	}
```

Cuadro de texto II.11. Constructor de la clase abstracta `com.cidero.upnp.AbstractService`

De entre las clases de Cidero que implementan servicios y que extienden a `AbstractService`, se modificaron las que corresponden con los servicios asociados a un *UPnP MediaRenderer*:

- **`com.cidero.upnp.AVTransport`**
- **`com.cidero.upnp.ConnectionManager`**
- **`com.cidero.upnp.RenderingControl`**

La modificación sólo consistió en la creación de un nuevo constructor, para cada una, que tuviese también este argumento de entrada, URL, y que llamase al constructor anterior de la clase base. A su vez, estos constructores serán llamados en los constructores de las respectivas clases de HomeAV que las extienden (`AVTransportService`, `ConnectionManagerService` y `RenderingControlService`).

```
/**
* Constructor (OSGI)
* @param device, servDescrptURL
* @throws InvalidDescriptionException
*/
public AVTransport( Device device, URL servDescrptURL ) throws
InvalidDescriptionException{
       super( device, servDescrptURL );
       logger.fine("Entered AVTransport base class constructor");
       logger.fine("Leaving AVTransport constructor");
}
```

Cuadro de texto II.12. Constructor de la clase `com.cidero.upnp.AVTransport`

```
/**
 * Constructor (OSGI)
 * @param device, servDescrptURL
 * @throws InvalidDescriptionException
 */
public ConnectionManager( Device device, URL servDescrptURL)
throws InvalidDescriptionException{
       super( device, servDescrptURL );
       logger.fine("Entered ConnectionManager base class constructor");
       logger.fine("Leaving ConnectionManager constructor");
}
```

Cuadro de texto II.13. Constructor de la clase `com.cidero.upnp.ConnectionManager`

```
/**
* Constructor (OSGI)
* @param device, servDescrptURL
* @throws InvalidDescriptionException
*/
public RenderingControl( Device device, URL servDescrptURL )throws
InvalidDescriptionException{
       super( device, servDescrptURL );
       logger.fine("Entered RenderingControl base class constructor");
       logger.fine("Leaving RenderingControl constructor");
}
```

Cuadro de texto II.14. Constructor de la clase `com.cidero.upnp.RenderingControl`

ANEXO III: Diagramas UML

Este anexo es un breve tutorial sobre UML para la descripción de los diagramas de este lenguaje que han sido utilizados en la elaboración del presente proyecto.

1. Introducción

El Lenguaje de Modelamiento Unificado (UML: *Unified Modeling Language*) es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. [UMLTUTORIAL]

A continuación se describe el formato de los tres tipos de diagramas UML que se pueden encontrar en varios apartados de la memoria:

- Diagrama de Casos de Uso.
- Diagrama de Clases.
- Diagrama de Secuencias.

2. Diagrama de Casos de Uso

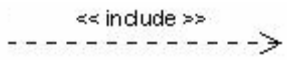
Este diagrama pretende representar de forma gráfica y sencilla la funcionalidad de un sistema y su interacción con los actores que intervienen. A continuación se describen las distintas entidades que figuran en el diagrama, así como sus posibles relaciones:

- Actores: son entidades externas al sistema pero que demandan o provocan una funcionalidad de este. Pueden se tanto personas como organizaciones o como otros sistemas.

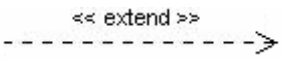
- Casos de uso: expresa una unidad coherente de funcionalidad del sistema. Es decir, algo que el sistema realiza.

- Relaciones entre los casos de uso:

  - Inclusión («include»): una instancia del Caso de Uso origen incluye también el comportamiento descrito por el Caso de Uso destino.



  - Extensión («extend»): El caso de uso origen extiende el comportamiento del caso de uso destino. Es una especificación de la funcionalidad que describe el caso de destino.



  - Herencia: El caso de uso origen hereda la especificación del caso de uso destino y posiblemente la modifica y/o amplía.

3. Diagrama de Clases

3.1 Introducción

Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contenimiento.

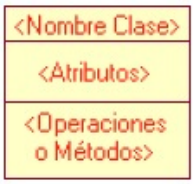
Un diagrama de clases esta compuesto por los siguientes elementos:

- Clases: atributos, métodos y visibilidad.
- Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

3.2 Clase

Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una clase). A través de ella podemos modelar el entorno en estudio (una Casa, un Auto, una Cuenta Corriente, etc.).

En UML, una clase es representada por un rectángulo que posee tres divisiones:



En donde:

- **Superior**: Contiene el nombre de la Clase
- **Intermedio**: Contiene los atributos (o variables de instancia) que caracterizan a la Clase (pueden ser private, protected o public).
- **Inferior**: Contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno (dependiendo de la visibilidad: private, protected o public).

Ejemplo:

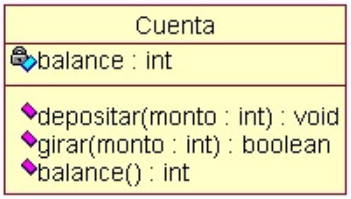
Una Cuenta Corriente que posee como característica:

- Balance

Puede realizar las operaciones de:

- Depositar
- Girar
- Balance

El diseño asociado es:



Atributos y Métodos de una Clase:

Atributos:

Los atributos o características de una Clase pueden ser de tres tipos, los que definen el grado de comunicación y visibilidad de ellos con el entorno, estos son:

- **public** (+): Indica que el atributo será visible tanto dentro como fuera de la clase, es decir, es accsesible desde todos lados.
- **private** (-): Indica que el atributo sólo será accesible desde dentro de la clase (sólo sus métodos lo pueden accesar).
- **protected** (#): Indica que el atributo no será accesible desde fuera de la clase, pero si podrá ser accesado por métodos de la clase además de las subclases que se deriven (ver herencia).

Métodos:

Los métodos u operaciones de una clase son la forma en como ésta interactúa con su entorno, éstos pueden tener las características:

- **public** (+): Indica que el método será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.
- **private** (-): Indica que el método sólo será accesible desde dentro de la clase (sólo otros métodos de la clase lo pueden accesar).
- **protected** (#): Indica que el método no será accesible desde fuera de la clase, pero si podrá serlo por métodos de la clase además de métodos de las subclases que se deriven.

3.3 Relaciones entre Clases:

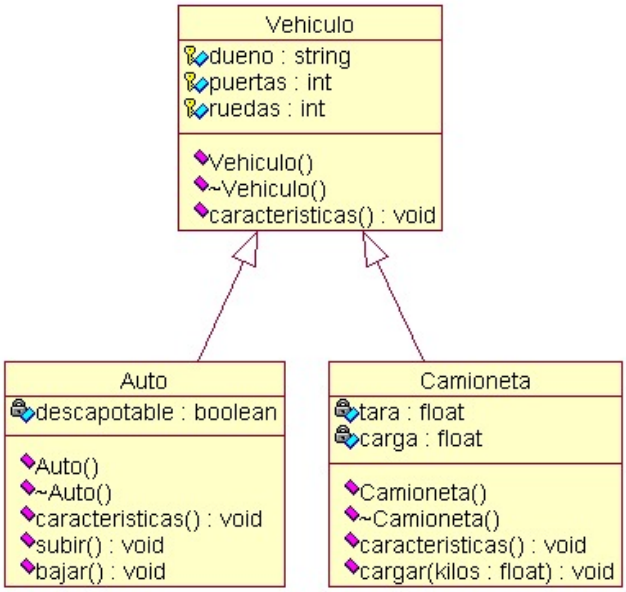
Ahora ya definido el concepto de Clase, es necesario explicar como se pueden interrelacionar dos o más clases (cada uno con características y objetivos diferentes).

Antes es necesario explicar el concepto de cardinalidad de relaciones: En UML, la cardinalidad de las relaciones indica el grado y nivel de dependencia, se anotan en cada extremo de la relación y éstas pueden ser:

- **uno o muchos**: 1..* (1..n)
- **0 o muchos**: 0..* (0..n)
- **número fijo**: m (m denota el número).

Herencia (Especialización/Generalización): 

Indica que una subclase hereda los métodos y atributos especificados por una Super Clase, por ende la Subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la Super Clase (public y protected), ejemplo:



En la figura se especifica que Auto y Camión heredan de Vehículo, es decir, Auto posee las Características de Vehículo (Precio, VelMax, etc) además posee algo particular que es Descapotable, en cambio Camión también hereda las características de Vehiculo (Precio, VelMax, etc.) pero posee como particularidad propia Acoplado, Tara y Carga.

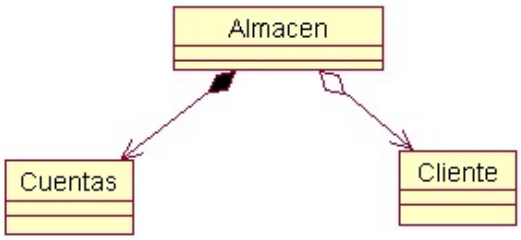
Cabe destacar que fuera de este entorno, lo único "visible" es el método Características aplicable a instancias de Vehículo, Auto y Camión, pues tiene definición publica, en cambio atributos como Descapotable no son visibles por ser privados.

Agregación: 

Para modelar objetos complejos, n bastan los tipos de datos básicos que proveen los lenguajes: enteros, reales y secuencias de caracteres. Cuando se requiere componer objetos que son instancias de clases definidas por el desarrollador de la aplicación, tenemos dos posibilidades:

- **Por Valor**: Es un tipo de relación estática, en donde el tiempo de vida del objeto incluido esta condicionado por el tiempo de vida del que lo incluye. Este tipo de relación es comúnmente llamada **Composición** (el Objeto base se construye a partir del objeto incluido, es decir, es "parte/todo").
- **Por Referencia**: Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye. Este tipo de relación es comúnmente llamada **Agregación** (el objeto base utiliza al incluido para su funcionamiento).

Un Ejemplo es el siguiente:



En donde se destaca que:

- Un Almacen posee Clientes y Cuentas (los rombos van en el objeto que posee las referencias).

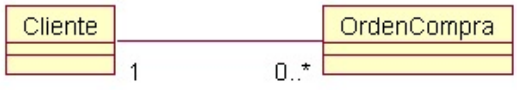
- Cuando se destruye el Objeto Almacen también son destruidos los objetos Cuenta asociados, en cambio no son afectados los objetos Cliente asociados.
- La composición (por Valor) se destaca por un rombo relleno.
- La agregación (por Referencia) se destaca por un rombo transparente.

La flecha en este tipo de relación indica la navegabilidad del objeto referenciado. Cuando no existe este tipo de particularidad la flecha se elimina.

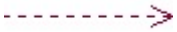
Asociación: 

La relación entre clases conocida como Asociación, permite asociar objetos que colaboran entre si. Cabe destacar que no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro.

Ejemplo:



Un cliente puede tener asociadas muchas Ordenes de Compra, en cambio una orden de compra solo puede tener asociado un cliente.

Dependencia o Instanciación (uso): 

Representa un tipo de relación muy particular, en la que una clase es instanciada (su instanciación es dependiente de otro objeto/clase). Se denota por una flecha punteada.

El uso más particular de este tipo de relación es para denotar la dependencia que tiene una clase de otra, como por ejemplo una aplicación grafica que instancia una ventana (la creación del Objeto Ventana esta condicionado a la instanciación proveniente desde el objeto Aplicacion):

Cabe destacar que el objeto creado (en este caso la Ventana gráfica) no se almacena dentro del objeto que lo crea (en este caso la Aplicación).

4. Diagrama de Secuencias

4.1 Introducción

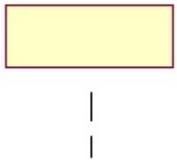
El diagrama de interacción, representa la forma en como un Cliente (Actor) u Objetos (Clases) se comunican entre si en petición a un evento. Esto implica recorrer toda la secuencia de llamadas, de donde se obtienen las responsabilidades claramente.

Dicho diagrama puede ser obtenido de dos partes, desde el Diagrama Estático de Clases o el de Casos de Uso (son diferentes).

Los componentes de un diagrama de interacción son:

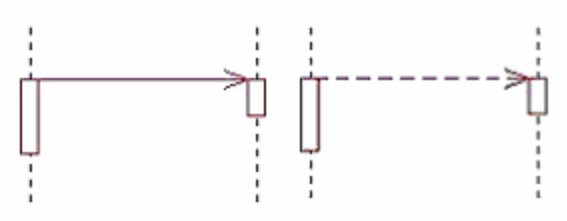
- Un Objeto o Actor.
- Mensaje de un objeto a otro objeto.
- Mensaje de un objeto a sí mismo.

4.2 Objeto/Actor:



El rectángulo representa una instancia de un Objeto en particular, y la línea punteada se utiliza para representar las llamadas a métodos del objeto de forma secuencial.

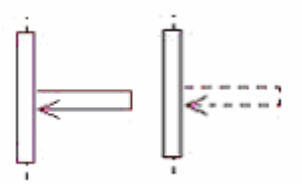
4.3 Mensaje a Otro Objeto:



Se representa por una flecha entre un objeto y otro, representa la llamada de un método (operación) de un objeto en particular. La línea discontinua indica el final de la ejecución del método. Esta línea aparece en el caso de que dentro del método se hagan llamadas a otros métodos, los cuales se muestran en el diagrama justo por debajo de su llamada y antes de ésta.

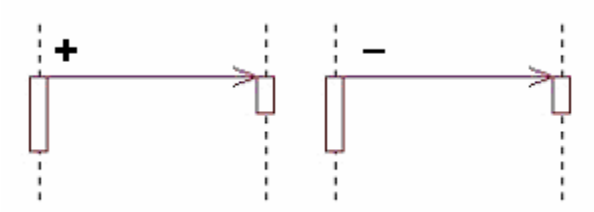
.

4.4 Mensaje al Mismo Objeto:



No solo se pueden realizar llamadas a métodos de objetos externos, también es posible visualizar llamadas a métodos desde el mismo objeto en estudio. Si el color de la línea es rojo, el método está definido como privado (sólo puede ser invocado dentro de la clase). De la misma manera que en el aparatado anterior, la línea continua indica el final de la ejecución del método.

En la siguiente figura se muestranlos símbolos “**+**” y “**-**”. El primero indica que el método está comprimido, es decir, invoca a otros métodos del análisis pero que no son mostrados. El siguiente indica que el método esta expandido, por lo que se muestran los métodos que invoca justo debajo y antes del símbolo correspondiente de final de método.



ANEXO IV: Ficheros de la aplicación

En este anexo se muestran los ficheros que son utilizados por la aplicación durante su ejecución.

1. Fichero de configuración

Se trata del fichero que utiliza HomeAV en su inicio para la configuración de algunos parámetros que modelan algunas funciones del mismo.

```
# HomeAV UPnP MediaRenderer

# This configuration file provides five parameters to configure some
# application's characteristics.

# sessionIPv6 -> enables|disables IPv6 network addressing
# seconRTPSession -> allows to create a second RTP session based on the first one.
#                    HomeAV create just one RTP session by default. This variable
#                    allows to create a second one with the following parameters:
#                     address = the same as the first RTP session
#                     port = [(the same as the first RTP session) + 2]
# interfaceNumber -> network interface identifier to get the IP address to be used
# nextAutoTransition -> enables|disables the 'setNextAVTransportURI' action
#                       to implement gapless multi-track playbacks
# cachingEnabled -> enables|disables to cache the resource to local disk during the
#                   playback

#-------------------------------------------------------------------------
# Allowed values:
# sessionIPv6 = no | yes
# secondRTPSession = no | yes
# interfaceNumber = 0 | 1 | ... (a positive integer)
# nextAutoTransition = yes | no
# cachingEnabled = yes | no
#-------------------------------------------------------------------------

sessionIPv6=no
secondRTPSession=no
interfaceNumber=0
nextAutoTransition=yes
cachingEnabled=yes
```

Cuadro de texto IV.1. Fichero de configuración de HomeAV (homeAV.config)

El significado de estos parámetros es:

- `sessionIPv6`: habilita o deshabilita el direccionamiento IPv6 para cuando la comunicación se realice en redes IPv6. Su valor por defecto es "*no*".
- `seconRTPSession`: para determinar si se desea o no que la aplicación cree una única sesión RTP para un único flujo, o bien, se creen dos sesiones para dos flujos. En tal caso, los parámetros de la segunda sesión serán los mismos que la primera (IP, puerto y ttl) excepto que el puerto será dos unidades superior al de la primera. Esta opción es indicada cuando se van a recibir dos flujos RTP, uno de audio y otro de video. Su valor por defecto es "*no*".
- `interfaceNumber`: este parámetro numérico identifica la interfaz de red que se desea utilizar para la comunicación. Su valor por defecto es "0", con el que se obtendrá la primera dirección IP válida de la primera interfaz de red habilitada.
- `nextAutoTransition`: con este parámetro se indica si se desea o no que esté disponible la acción `SetNextAVTransportURI()` del servicio *AVTransport Service*. Gracias a ésta HomeAV puede preparar la siguiente reproducción para que cuando acabe la que esté en proceso no haya espacios en blanco entre ambas. Sólo utilizada en caso de reproducciones de contenido de audio y/o video obtenido mediante HTTP. Su valor por defecto "*yes*".
- `cachingEnabled`: habilita o deshabilita la opción de *caching* del `Player`, por la cual, un fichero puede ser descargado localmente y de forma temporal previamente a su reproducción. Esta opción es utilizada en el proceso de reproducción de contenido de audio y/o video obtenido mediante HTTP.

2. Documento descriptor del dispositivo UPnP

A continuación se muestra el documento XML de descripción del dispositivo HomeAV. Por su extensión, no se muestran los correspondientes descriptores de los servicios contenidos, *RenderinControl Service*, *ConnectionManager Service* y *AVTransport Service*. La información sobre las acciones implementadas se mencionan en la descripción sobre las clases Java correspondientes descritas en el apartado sobre HomeAV en el capitulo “Implementación”.

Todos estos documentos serán solicitados por el *AV Control Point* para crear el modelo del dispositivo *UPnP MediaRenderer* (HomeAV), pero también son utilizados por la propia aplicación para obtener, entre otras cosas, la lista de servicios contenidos, así como la lista de sus correspondientes acciones o variables de estado definidas.

Documento XML de descripción del dispositivo HomeAV (DCPD):

```
<?xml version="1.0" encoding="utf-8"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
   <specVersion>
      <major>1</major>
      <minor>0</minor>
   </specVersion>
   <device>
viceType>urn:schemas-upnp-org:device:MediaRenderer:1</deviceType>
      <friendlyName></friendlyName>
      <manufacturer></manufacturer>
      <manufacturerURL></manufacturerURL>
      <modelDescription>Renders audio video files and images from UPnP
MediaServers</modelDescription>
      <modelName>HomeAV UPnP Renderer</modelName>
      <modelNumber>1.0</modelNumber>
      <modelURL></modelURL>
      <UDN></UDN>
      <serviceList>
         <service>
            <serviceType>urn:schemas-upnp-org:service:AVTransport:1</serviceType>
            <serviceId>urn:upnp-org:serviceId:urn:schemas-upnp-
org:service:AVTransport</serviceId>
            <SCPDURL>/service/AVTransport.xml</SCPDURL>
            <controlURL>/service/AVTransport_control</controlURL>
            <eventSubURL>/service/AVTransport_event</eventSubURL>
         </service>
         <service>
            <serviceType>urn:schemas-upnp-org:service:RenderingControl:1</serviceType>
            <serviceId>urn:upnp-org:serviceId:urn:schemas-upnp-
org:service:RenderingControl</serviceId>
            <SCPDURL>/service/RenderingControl.xml</SCPDURL>
            <controlURL>/service/RenderingControl_control</controlURL>
            <eventSubURL>/service/RenderingControl_event</eventSubURL>
         </service>
         <service>
            <serviceType>urn:schemas-upnp-org:service:ConnectionManager:1</serviceType>
            <serviceId>urn:upnp-org:serviceId:urn:schemas-upnp-
org:service:ConnectionManager</serviceId>
            <SCPDURL>/service/ConnectionManager.xml</SCPDURL>
            <controlURL>/service/ConnectionManager_control</controlURL>
            <eventSubURL>/service/ConnectionManager_event</eventSubURL>
         </service>
      </serviceList>
   </device>
</root>
```

Cuadro de texto IV.2. Documento XML de descripción del dispositivo HomeAV

ANEXO V: Instrucciones de instalación

En este anexo se enumeran los pasos necesarios pasta la puesta en marcha de la aplicación.

Es necesario advertir que esta aplicación no fue desarrollada para su distribución ni comercialización, por lo que no se han considerado los aspectos restrictivos al respecto que marca la licencia SCSL de Sun Microsystems [SUNSCSL] para JMF.

1. Requerimientos del sistema

A continuación se enumeran los requerimientos previos del sistema y en base a los cuales la aplicación ha sido desarrollada y probada:

- Hardware: PC con un mínimo de 256MB de RAM y procesador de 800MHz.
- Sistemas Operativos: plataformas Windows y Linux de 32 bits. Los sistemas operativos utilizados han sido: Windows XP y Linux Ubuntu 8.04.
- UPnP: debe estar habilitado UPnP en el sistema operativo y configurado el cortafuegos para que permita su uso.
- JVM: la aplicación está programada en Java y ha sido empaquetada en un *bundle*, creando un fichero JAR (Java ARChive), por lo que se hace necesario tener instalada una JVM (Máquina Virtual Java). Las versiones utilizadas son:
  - JDK 1.6 para Windows [JDK1.6].
  - JDK 1.7 para Linux [JDK1.7].
- Implementación OSGi: una vez instalada la JVM, se ha de obtener una implementación del *Framework* de OSGi. Las que se muestran a continuación (de software libre) son sobre las que la aplicación ha sido probada:
  - Equinox 3.3.0 de Eclipse [EQUINOX]
  - Felix 1.8.0 de Apache [FELIX]

2. Requerimientos de la aplicación

Sin JMF instalada en el equipo:

- Copiar el fichero de propiedades de JMF, *jmf.properties*, que se distribuye con el *bundle* en el mismo directorio que el ".jar" de la implementación OSGi correspondiente.
- Para plataformas Linux, se hace necesario exportar la variable del sistema *LD_LIBRARY_PATH* añadiendo la ruta del directorio "natives", con las librerías nativas de JMF, que se distribuye con el *bundle*. Otra solución, sería copiar estas librerías al directorio donde se encuentren las librerías del sistema, normalmente "usr/lib".

Con JMF instalada en el equipo:

- Se han de instalar los *plug-ins* de *codecs* de los que hace uso la aplicación:
  - Fobs4JMF v0.4.1 [FOBS]
  - MP3 *plug-in* de Sun [MP3PLUGIN]
  - JFFMPEG v 1.1.0[JFFMPEG]

  Para este proceso seguir los pasos que se indican en las respectivas webs, con un matiz respecto de las instrucciones para Jffmpeg: obviar las referencias a las librerías nativas ".so" y ".dll".
- Tras esto, y una vez añadido los *plug-in* al registro, sustituir el *jmf.properties* contenido en el directorio donde se encuentra el ".jar" de la librería JMF, y sustituirlo por el que se distribuye con el *bundle*.

Una vez realizado esto, ya se puede controlar el ciclo de vida del *bundle* en el *Framework* OSGi por medio de los correspondientes comandos `install`, `start`, `stop` y `uninstall`.

ANEXO VI: Presupuesto

Este anexo contiene una breve descripción de las fases de ejecución del proyecto y una memoria resumida de los costes y gastos generados de la realización del mismo.

1. Fases del proyecto

En las próximas líneas se enumeran las tareas que se han realizado agrupadas en las distintas fases que definen la consecución de este proyecto:

Fase 1: Estudios previos: Marco Teórico

En esta fase se adquirieron los conocimientos teóricos necesarios para el desarrollo de la aplicación. Las tareas realizadas son:

- Estudio de la tecnología UPnP.
- Estudio de la Arquitectura AV UPnP, con especial atención al *UPnP MediaRenderer*.
- Análisis de las soluciones de *UPnP MediaRenderer* disponibles en el mercado actual.
- Estudio de la librería multimedia JMF para su utilización en el tratamiento del contenido multimedia.
- Análisis del paquete Java AWT y Java Swing para la creación de interfaces gráficas y visualización de imágenes.
- Estudio de la tecnología OSGi y el API que define la plataforma.

Duración = 320 horas.

Fase 2: Diseño del prototipo

En esta fase se realizó la implementación propiamente dicha del dispositivo *UPnP MediaRenderer* en base a los conocimientos adquiridos en la fase previa. Las tareas que se definen suponen la implementación de cada una de las unidades funcionales de la aplicación. Estas son:

- Análisis de requisitos del sistema y definición de las herramientas a utilizar.
- Diseño e Implementación del dispositivo UPnP y la definición de los servicios asociados a un *UPnP MediaRenderer*.
- Modificación de las librerías de Cidero/Cyberlink.
- Diseño e implementación de la funcionalidad para la reproducción de contenido de audio y video obtenido mediante el protocolo HTTP.
- Diseño e implementación de la funcionalidad para la visualización de imágenes.
- Diseño e implementación de la funcionalidad para la reproducción de flujos RTP de audio y video.
- Integración de la aplicación en un *bundle* OSGi.

Duración = 410 horas.

Fase 3: Pruebas del prototipo:

Una vez obtenido el prototipo de *UPnP MediaRenderer* integrado en OSGi, se realizó una fase de pruebas para comprobar el correcto funcionamiento de la aplicación en los posibles escenarios.

- Pruebas de funcionalidad en las que se comprueba la respuesta de la aplicación en un escenario real.
- Pruebas en escenarios IPv6 para comprobar la compatibilidad con este protocolo de red.
- Pruebas en escenarios con más de un *Control Point*.

- Análisis de los tiempos de respuesta de la aplicación para la fase de descubrimiento UPnP y la reproducción de ficheros de audio y video.

Duración = 65 horas.

Fase 4: Elaboración de la documentación.

Duración = 280 horas.

2. Costes

Para la elaboración del presupuesto se han considerado aquellos costes que están relacionados con la utilización de bienes materiales y los que provienen de la mano de obra para la realización del proyecto.

Costes materiales

Son los derivados de la creación del entorno de trabajo donde se ha desarrollado y probado la aplicación, incluyendo los costes indirectos estimados por el consumo energético, material fungible y comunicaciones.

Se han considerado únicamente los equipos que han sido necesarios en este proceso aunque se hayan utilizado un número mayor para las pruebas.

Concepto	Precio (€)	Periodo de Amortización (meses)	Periodo de uso (meses)	Importe (€)
2 Ordenadores Personales (Windows XP HE/Prof)	1.400	36	10	388,89
1 Ordenador Personal (Linux Ubuntu)	500	36	10	138,89
1 Router (Wi-Fi + 4 puertos Ethernet)	60	36	10	16,67
Costes indirectos				460
TOTAL				**1.004,45**

Tabla VI.1. Costes materiales

Costes de mano de obra:

Cumpliendo una normativa europea, los colegios profesionales han dejado de publicar baremos orientativos de los honorarios. Como consecuencia, la cifra utilizada que define el sueldo por hora de un Ingeniero Técnico de Telecomunicaciones es una estimación deducida de la información obtenida en webs de empleo [WEBEMPLEO].

En este proyecto ha intervenido un único Ingeniero Técnico de Telecomunicaciones para realizar el conjunto de tareas definidas en las distintas fases del proyecto. Por tanto, el coste derivado de la mano de obra es:

Responsable	**Horas**	**€/Hora**	**Importe (€)**
Ingeniero Técnico de Telecomunicaciones	1075	40	**43.000**

Tabla VI.2. Costes de mano de obra

Presupuesto total:

En la siguiente tabla se resume el presupuesto total para la realización del proyecto:

Concepto	**Importe (€)**
Costes materiales	1.004,45
Costes de mano de obra	43.000
Total base imponible	44.004,45
IVA (16%)	7.040,71
TOTAL	**51.045,16**

Tabla VI.3. Presupuesto total

ANEXO VII: Bibliografía

[AUNADOC]: Documento de la fundación AUNA sobre tecnologías para interconexión de *Home Networks*
http://www.fundacionorange.es/areas/historico/pdf/4.pdf

[BARKIDSUPM]: Documento de presentación del proyecto BarkIDS realizado por la Universidad Politécnica de Madrid
http://internetng.dit.upm.es/Barkids.pdf

[CIDERO]: Sitio web oficial del proyecto Cidero
http://www.cidero.com/

[CYBERGARAGE]: Sitio web oficial del dominio CyberGarage
http://www.cybergarage.org/

[COITTDOC]: Nota informativa del COITT (Colegio Oficial de Ingenieros Técnicos de Telecomunicaciones) sobre el baremo de honorarios
http://www.coitt.es/res/libredocs/Honorarios.pdf

[COMPERE]: Página web de la aplicación Compère
https://coherence.beebits.net/wiki/Coherence*MediaRenderer*

[ECLIPSE]: Sitio web oficial del proyecto Eclipse
http://www.eclipse.org/

[EQUINOX]: Sitio web oficial del proyecto Equinox
http://www.eclipse.org/equinox/

[EXPEDITOR]: Página web oficial de IBM para Lotus Expeditor
http://www-01.ibm.com/software/lotus/products/expeditor/

[FELIX]: Sitio web del proyecto Felix
http://felix.apache.org/site/index.html

[FMJ]: Sitio web oficial del proyecto FMJ
http://fmj-sf.net/index.php

[FOBS]: Sitio web oficial del proyecto FOBS de Omnividea (*codec plug-in* JMF)
http://fobs.sourceforge.net/

[GMRENDER]: Sitio web oficial de la aplicación GMediaRender
http://gmrender.nongnu.org/

[JAVAFX]:
Sitio web oficial de Sun Microsystems para JavaFx
http://www.sun.com/software/javafx/
Documento JavaFx y JMC redactado por la conferencia JavaOne de Sun Microsystems
http://developers.sun.com/learning/javaoneonline/2008/pdf/TS-6509.pdf
Documento JavaFx, JMC y estado actual de la librería JMF redactado por la conferencia JavaOne de Sun Microsystems
http://developers.sun.com/learning/javaoneonline/2008/pdf/TS-7372.pdf

[JDK1.6]: Sitio web oficial de Sun Microsystems para la versión 6 de la Java SE
http://java.sun.com/javase/6/

[JDK1.7]: Sitio web sobre el proyecto de desarrollo de la versión 7 de la Java SE
https://jdk7.dev.java.net/

[JFFMPEG]: Sitio web oficial del proyecto Jffmepg (*codec plug-in* JMF)
http://jffmpeg.sourceforge.net/

[JMC]:
Documento JavaFx y JMC redactado por la conferencia JavaOne de Sun Microsystems
http://developers.sun.com/learning/javaoneonline/2008/pdf/TS-6509.pdf
Documento JavaFx, JMC y estado actual de la librería JMF redactado por la conferencia JavaOne de Sun Microsystems
http://developers.sun.com/learning/javaoneonline/2008/pdf/TS-7372.pdf

[JMF]: Sitio web oficial de Sun Microsystems para JMF
http://java.sun.com/javase/technologies/desktop/media/jmf/index.jsp

[JMFCUSTOMRTP]: Página web de Sun Microsystems para la solución RTPConnector
http://java.sun.com/javase/technologies/desktop/media/jmf/2.1.1/solutions/RTPConnector.html

[JMFFORO]: Sección para JMF del Foro de Sun Microsystems
http://forums.sun.com/forum.jspa?forumID=28&start=0

[JMFGUIA]: Guía oficial del API de JMF
http://java.sun.com/javase/technologies/desktop/media/jmf/2.1.1/guide/index.html

[JMFOUT]: Documento JavaFx, JMC y estado actual de la librería JMF redactado por la conferencia JavaOne de Sun Microsystems
http://developers.sun.com/learning/javaoneonline/2008/pdf/TS-7372.pdf

[JVLC]:
Sitio web oficial del proyecto VideoLan para JVLC
https://trac.videolan.org/jvlc/
Sitio web del proyecto JVLC
http://jvlc.ihack.it/maven-site/jvlc-core/project-info.html

[KNOPFLERFISH]: Sitio web del proyecto Knopflerfish
http://www.knopflerfish.org/

[MAINTAINJ]: Sitio web oficial de la aplicación MaintainJ
http://www.maintainj.com/

[MPLAYER]: Sitio web oficial de la aplicación MPlayer
http://www.mplayerhq.hu/design7/info-es.html

[MP3PLUG-IN]: Página web de Sun Microsystems para el MP3 *codec plug-in* de JMF
http://java.sun.com/javase/technologies/desktop/media/jmf/mp3/download.html

[NERO]: Página web de la aplicación Nero9
http://www.nero.com/esp/nero9-introduction.html

[NEWTON]: Sitio web oficial del proyecto Newton
http://newton.*codec*auldron.org

[OSGi]: Sitio web oficial de OSGi
http://www.osgi.org

[OSGiAPUNTES]: Apuntes de la asignatura “Presentación Multiplataforma” del tercer curso de Ingeniería Técnica de Telecomunicaciones (esp. Sonido e Imagen) de la Universidad Carlos III de Madrid (curso 2007/2008)

[OSGiCASADOMO]: Documento informativo sobre OSGi de la revista digital CASADOMO
http://www.casadomo.com/noticiasDetalle.aspx?c=49&m=15&idm=60&pat=14&n2=14

[OSGiSERVICES]: Sección del sitio oficial de OSGi sobre la tecnología
http://www.osgi.org/About/Technology#Standard_Services

[OSGiSPECIFIC]: Página web del sitio oficial de OSGi para la descarga de los documentos de la especificación 4 de la plataforma
http://www.osgi.org/Release4/Download

[POCKETPLAYER]: Página web de la aplicación Pocket Player 4
http://www.conduits.com/products/player/

[POWERCINEMA]: Página web de la aplicación PowerCinema 6
http://es.cyberlink.com/products/powercinema/radio-music_es_ES.html

[PROSYST]: Sitio web oficial del proyecto PROSYST
http://www.prosyst.com

[SDE]: Página web oficial para la aplicación Smart Development Environment (SDE)
http://www.visual-paradigm.com/product/sde/

[SDKPLATINUM]: Sitio web de Platinum (UPnP SDK)
http://www.plutinosoft.com/blog/projects/platinum/

[SDKLIBUPNP]: Página web de “libupnp” (UPnP SDK)
http://pupnp.sourceforge.net/

[SUNSCSL]: Licencia SCSL de Sun Microsystems
http://java.sun.com/j2se/1.5.0/scsl_5.0-license.txt

[UMLTUTORIAL]: Tutorial web sobre UML
http://www.dcc.uchile.cl/~psalinas/uml/introduccion.html

[UPNPAV]: Sección del sitio web oficial del Foro UPnP para la arquitectura UPnP AV Architecture
http://www.upnp.org/specs/av/

[UPNPCASADOMO]: Documento informativo sobre UPnP de la revista digital CASADOMO
http://www.casadomo.com/noticiasDetalle.aspx?c=25&m=29&idm=32&pat=148&n2=148

[UPNPDEVARCH]: Documento técnico sobre la arquitectura de dispositivo de UPnP
http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

[UPNPEINDHOVEN]: Documento sobre UPnP de la universidad de Eindhoven *University of Technology*
http://www.win.tue.nl/ipa/archive/springdays2002/ThangSlides.ppt

[UPNPFORUM]: Sitio web oficial del Foro UPnP
http://www.upnp.org/

[UPNPMUSE]: Documento sobre UPnP titulado “*The Home Entertainment System based on UPnP AV Framework*” de la universidad Kyungpook*National University*.

[UPNPPROT]: Documento técnico sobre la tecnología UPnP realizado por Microsoft
http://www.upnp.org/download/UPNP_UnderstandingUPNP.doc

[UPNPQOS]: Sección del sitio oficial del Foro UPnP sobre la especificación para la calidad de servicio (QoS)
http://www.upnp.org/specs/qos/

[UPNPSECURITY]: Sección del sitio oficial del Foro UPnP sobre la especificación para seguridad
http://www.upnp.org/standardizeddcps/security.asp

[VLC]: Sitio web oficial de VideoLan (aplicación VLC)
http://www.videolan.org/vlc/

[WEBEMPLEO]: Sitio web de empleo para autónomos
http://www.infolancer.net/freelancers/

[WMP11: Sitio web oficial de Microsoft para la aplicación Windows Media Player 11
http://www.microsoft.com/windows/windowsmedia/es/player/11/default.aspx

[XBMC]: Sitio web oficial de XBMC Media Center
http://xbmc.org/home/

ANEXO VIII: Glosario

ADPCM	Modulación por codificación de impulsos diferencial adaptativa (*Adaptive Differential Pulse Code Modulation*)
API	Interfaz de programación de aplicaciones (*Application Programming Interface*)
AVI	Entrelazado de audio y video (*Audio Video Interleave*)
AWT	Herramientas Abstractas de Ventana (*Abstract Window Toolkit*)
bps	Bits por segundo
CD	Disco compacto (*Compact Disc*)
CPU	Unidad central de procesamiento (*Central Processing Unit*)
DCP	Protocolo de control de dispositivos (*Device Control Protocol*)
DCPD	Documento del protocolo de control de dispositivos (*Device Control Protocol Document*)
CVM	Máquina virtual compacta (*Compact Virtual Machine*)
DHCP	Protocolo de configuración dinámica de equipos (*Dynamic Host Configuration Protocol*)
DNS	Sistemas de nombres de dominio (*Domain Name System*)
DVD	Disco digital versátil (*Digital Versatile Disc*)
DVI	Interfaz visual digital (*Digital Visual Interface*)
FMJ	Libertad para el multimedia en Java (*Freedom for Media in Java*)
GENA	Arquitectura de notificación de eventos generales (*General Event Notification Architecture*)
GIF	Formato de intercambio de gráficos (*Graphics Interchange Format*)
GNU	GNU no es Unix (*GNU's Not Unix*) (acrónimo recursivo)
GSM	Sistema global para comunicaciones móviles (*Global System for Mobile communications*)
GUI	Interfaz gráfica de usuario (*Graphical User Interface*)
HAVi	Interoperabilidad audiovisual en el hogar (*Home Audio Video Interoperability*)
HomePNA	Alianza para la redes telefónicas del hogar (*Home Phoneline Networking Alliance*)
HTML	Lenguaje de marcación de hipertexto (*HyperText Markup Language*)
HTTP	Protocolo de transferencia de hipertexto (*HyperText Transfer Protocol*)
HTTPMU	Transmisión múltiple HTTP (*HTTP Multicast*)
HTTPU	Transmisión única HTTP (*HTTP Unicast*)
IANA	Autoridad para la asignación de números de Internet (*Internet Assigned Numbers Authority*)
ID	Identificador (*IDentifier*)
IDE	Entorno de desarrollo integrado (*Integrated Development Environment*)
IMA	Multiplexado inverso sobre ATM (*Inverse Multiplexing over ATM*)
IP	Protocolo de Internet (*Internet Protocol*)
IPv4	Versión 4 del protocolo de Internet (*IP version 4*)
IPv6	Versión 6 del protocolo de Internet (*IP version 6*)
JAR	Archivo Java (*Java ARchive*)
JDK	Kit de desarrollo Java (*Java Development Kit*)
JINI	Infraestructura de red inteligente Java (*Java Intelligent Network Infraestructure*)
JMC	Componente Multimedia de Java (*Java Media Component*)
JMF	Entorno de trabajo multimedia Java (*Java Multimedia Framework*)
JNA	Acceso nativo de Java (*Java Native Access*)
JPEG	Grupo de expertos en empalme de imágenes (*Joint Photographers Experts Group*)
JVM	Máquina virtual Java (*Java Virtual Machine*)
JVLC	*Java VLC*
J2ME	Java 2 edición micro (*Java 2 Micro Edition*)
J2SE	Java 2 edición estándar (*Java 2 Standard Edition*)
KVM	Máquina virtual basada en el kernel (*Kernel-based Virtual Machine*)
LCD	Pantalla de cristal líquido (*Liquid Crystal* Display)
MPEG	Grupo de expertos en imágenes en movimiento (*Moving Picture Experts Group*)
MP3	MPEG-1 capa 3 (*MPEG-1 Layer 3*)
OSGi	Iniciativa de pasarela para sistemas abiertos (*Open System Gateway Initiative*)

PC Ordenador personal (*Personal Computer*)
PCM Modulación por impulsos modificados (*Pulse Code Modulation*)
PDA Asistente Digital Personal (*Personal Digital Assistant*)
PNG Gráficos portátiles de red (*Portable Network Graphics*)
RAM Memoria de acceso aleatorio (*Random Access Memory*)
RGB Rojo Verde Azul (*Red Green Blue*)
RPC Llamada de procedimiento remoto (*Remote Procedure Call*)
RTP Protocolo de transporte en tiempo real (*Real-Time Transport Protocol*)
RTCP Protocolo de control en tiempo real (*Real Time Control Protocol*)
RTSP Protocolo de transferencia de flujos en tiempo real (*Real Time Streaming Protocol*)
SCPD Definición del protocolo de control de servicio (*Service Control Protocol Definition*)
SCSL Licencia para código fuente de la comunidad Sun (*Sun Community Source License*)
SDK Kit de desarrollo estándar (*Standard Development Kit*)
SID Identificador de la subscripción (*Subscription IDentifier*)
SOAP Protocolo de acceso sencillo a objetos (*Simple Object Access Protocol*)
SSDP Protocolo de descubrimiento sencillo de servicios (*Simple Service Discovery Protocol*)
SSL Capa de seguridad del *socket* (*Secure Socket Layer*)
SUN Red universitaria de Standford (*Stanford University Network*)
TCP Protocolo de control de transmisión (*Transmission Control Protocol*)
TIFF Formato de archivo de imágenes con etiquetas (*Tagged Image File Format*)
TTL Tiempo de vida (*Time-To-Live*)
UDP Protocolo de datagramas de usuario (*User Datagram Protocol*)
UML Lenguaje unificado de modelado (*Unified Model Language*)
UPC Código de producto universal (*Universal Product Code*)
UPnP Conectar y usar universal (*Universal Plug and Play*)
URI Identificador de recurso uniforme (*Uniform Resource Identifier*)
URL Localizador de recurso uniforme (*Uniform Resource Locator*)
URN Nombre de recurso uniforme (*Uniform Resource Name*)
USB Bus serie universal (*Universal Serial Bus*)
USN Nombre de serie único (*Unique Serial Name*)
UUID Identificador único universal (*Universal Unique IDentifier*)
VLC Cliente VideoLan (*VideoLan Client*)
WAV Formato de audio de forma de onda (*Waveform Audio Format*)
XML Lenguaje de marcado extensible (*eXtended Markup Language*)
XP Programación extrema (*Extreme Programming*)